Word of thanks

I would like to thanks several people who have made this project possible. First of to all the managers of Kivnon Logística S.L. Luis Gómez, Juan Prieto and Xavier Munells for letting me work on this project in the company. To Pere Ponsa, my thesis advisor, who has helped and suggested in many aspects of the project. To Sergi Miñambres for his technical assistance while learning to code in C#. To Oscar Mendez who has let me use pictures of him to show different gestures. And finally to all who have participated in the gesture database construction and in the tests.

Abstract

Interaction between humans and AGVs (Automatic guided vehicles) is often reduced to non-existent or just some pushbuttons, touchscreens or signalling lamps [17][18][19][20]. In recent years some researchers have attempted to improve the interaction through AGVs that follow people [53], path planning through gestures or robots that are able to ask for help if they cannot carry out a task on its own. [54] This project proposes extending the HRI (Human Robot Interaction) of Kivnon Logística's AGVs with gesture recognition and human localization through depth cameras and with emotion-simulating feedback through a screen. The aim for this technology introduction is to develop a shopping cart like AGV that follows people on gesture commands. The developed system will be built on Kivnon Logística's AGV model K11 mouse as a prototype. Later on a specific AGV will be designed. The final aim of this project is to introduce smart interactive AGVs in industrial environments focused in production areas.

Keyword: AGV, HRI, gesture recognition, emotion-imitating feedback, depth cameras.

Index

Figure's Index

Tables

Algorithms

CHAPTER 1: INTRODUCTION

1.1 Summary

Interaction with industrial systems has been limited for a long time to pushbuttons and light signals. In the past few years touchscreens have been introduced displaying SCADAs on them enabling more complex orders in an easier way. One step further would be the use of human motion as input. The videogame industry released in 2010 the motion sensing input device Kinect providing a software development kit for game developers. Soon enough the robotic industry found use for it porting it to ROS (Robot Operating System).

Kivnon Logísitica has encountered in several occasions operators who complained about unnatural interaction with their AGVs. Thus they have considered implementing natural gesture recognition using the Kinect platform.

Natural gesture are those gestures humans use in everyday life to communicate with other humans; for example the "come closer" or "stop" indications used while helping a driver to park his car. This type of body language presents an important drawback: each person executes gestures differently and gestures with different meaning could be very similar. Some authors even declare that "natural gestures" are in fact not natural. [39] Don Norman states that gestures may vary its meaning through each culture. For example the Indian head shake for "no" is the western head shake for "yes" and vice versa. For this reason this project will use European style gestures.

Authors have faced the gesture recognition problem with mainly 2 approaches. Jie Yang and Yangsheng Xu proposed multi-dimensional Hidden Markov Models (HMM) in 1994 obtaining 99.78% accuracy for an isolated recognition task with nine gestures. [1]. K. Takahashi et. al. proposed the use of the Dynamic Time Warping (DTW) algorithm in 1993 [2]. But it was not until the release of the Kinect sensor that DTW gained popularity amongst the gesture recognition researcher.

Kivnon Logística has decided to use the DTW algorithm due to its simplicity and still good performance. However basic DTW algorithm is not functional; it is computational expensive as it has to compare multidimensional gestures to a database that can contain hundreds of gesture models. Therefore this project has focused heavily on time consuming optimizations based on bounding limits [6] and local constraints [10][21].

1.2 Introduction to AGVs

There exist several definitions for AGVs:

- An automated guided vehicle or automatic guided vehicle (AGV) is a mobile robot that follows markers or wires in the floor, or uses vision or lasers. They are most often used in industrial applications to move materials around a manufacturing facility or a warehouse. (Wikipedia) [22]
- An automatic guided vehicle system (AGVS) consists of one or more computer-controlled, wheel-based load carriers (normally battery powered) that runs on the plant or warehouse floor (or if outdoors on a paved area) without the need for an on-board operator or driver. (HMI) [23]
- Automated Guided Vehicles (AGVs) are mobile robots for use in production, warehousing or any environment that requires the routine movement of products. A typical use for AGVs are as forklift replacements, instead of a forklift and driver an AGV in communication with a control server will pick up a load from one position and unload it in another. (NDC Automation) [24]
- The term "automated guided vehicle" (AGV) is a general one that encompasses all transport systems capable of functioning without driver operation. The term "driverless" is often used in the context of automatic guided vehicles to describe industrial trucks, used primarily in manufacturing and distribution settings, which would conventionally have been driver-operated. (USLegal.com) [25]
- A materials handling system that uses automated vehicles such as carts, pallets or trays which are programmed to move between different manufacturing and warehouse stations without a driver. These systems are used to increase efficiency, decrease damage to goods and reduce overhead by limiting the number of employees required to complete the job. (BusinessDictionary.com) [26]

To put it simple: AGVs are “driverless vehicles used to move material efficiently in a facility”. [27] The main benefits of the use of AGVs are:

- Reduce manufacturing costs
- Employees can be reassigned to areas where they can add value to the production
- Safe and efficient material movement
- Operates continuously, does not need breaks
- Can work in hazardous or cold environments

Over the past 50 years AGVs have developed into several types depending on their use. They can be classified in:

- Forked vehicles: These types of AGVs resemble typical manual lift trucks. They can operate material not only on ground level but also on higher levels enabling it to interact with material on shelves.

Figure 1: Forked vehicle manipulating pallets on different height levels

- Unit load vehicles: Compact AGV that carries load on their top platform. Loading is usually performed via conveyor or a lift. Some unit load vehicles are able to elevate their top platform making it possible to lift carts. Other ones have a retractile pin instead of a platform to hook up carts.

Figure 2: Unit load vehicle with retractile pin pulling a cart.

- Tuggers: Usually compact AGVs that tow several wheeled carts. Load is usually placed manually on the towed carts.

Figure 3: Tugger vehicle towing 3 wheeled carts.

- Heavy Burden Carriers: Large wheeled platforms that are able to carry extreme heavy loads.

Figure 4: Heavy burden carrier transporting plane parts.

- Custom: Sometimes there is a need of a specific AGV form. For this reason manufacturer may design AGVs adapted to the size and shape of the load or adapted to the functionality or environment.

1.3 Motivation

The author of this project has worked for more than 3 years on the AGV sector developing human-robot-interaction and environment-robot-interaction devices working directly with clients, thus has obtained a lot of experience and feedback. This has led him to consider to condense all obtained knowledge into an improved and natural interface to interact with AGVs.

1.4 Objectives

The main objective of this project is to develop and implement a gesture recognition system to control AGVs in a natural way. The secondary objectives are:

- Increase the liability of this gesture recognition system.
- Implement a natural feedback system.
- Create an AGV to display in fairs the state of the art of AGV's HRI.

In order to achieve these objectives the developed system needs to accomplish the following goals:

- Detect the user and ignore the rest of the humans entering the field of vision.
- Analyse the user's body to detect and interpret gestures.
- Follow the user at a safe distance.
- Search for the user if he exits the field of vision.

1.5 Structure

This project is divided in 19 chapters:

- Introduction: the present chapter. This chapter intends to introduce the reader into the project and the AGV industry.
- The company's AGV: This chapter presents the AGV used in this project. It explains all the interaction possibilities with other AGVs, environment and humans.
- Development Platform: Here the different hardware and software involved in the project will be explained.
- Data pre-process: This chapter focuses on the filtering of data and reference transformations of the sensor's data.
- Gesture recognition algorithm: The basic gesture recognition algorithm is explained here. In addition, enhancements of this algorithm are described and discussed.
- User recognition: Different methods to recognize users are discussed here.
- AGV Navigation: This chapter explains 2 navigation methods and how gesture recognition is applied to each of them. In addition it explains the velocity controller's algorithm.
- Gestures: All implemented gestures are presented and explained here.

- Virtual Assistant: This chapter discuses the reasons for using an avatar and its design. It explains also every different "emotion" of the avatar.
- GRAFCET representation: This chapter introduces the reader into GRAFCET theory and explains the operations the system performs.
- Security: This chapter explains how to use the GEMMA Guide and how it is applied in the project.
- Assembly: This chapter deals with the rest of topics needed to build the project that were not explained on other chapters.
- Result and conclusions: Here a test is carried out to measure the performance of the gesture recognition system. Results are shown and analysed.
- Further work: Aspects that were not introduced in this project but will be implemented in the near future are discussed here.
- Project cost: This chapter explains the cost of development of this project.
- Glossary: A small glossary explaining the key words used in this project.
- References: Bibliography referenced in the project.
- Additional bibliography: Bibliography consulted to develop this project but not referenced.
- Annex: Further information about the project: Formulae, schematics, PCB layouts and codes.

CHAPTER 2: THE COMPANY'S AGV

This project is conceived to improve the AGVs of the company Kivnon Logística, therefore this chapter will introduce them and their systems.

2.1 Kivnon Logística

In the year 2010 the engineering company DCLane started to create its own AGVs for their logistic projects. In 2011 the AGV design department was separated from DCLane and Kivnon Logística was founded. The main aim was to design and manufacture AGVs that satisfied DCLane's projects' needs. As Kivnon Logística focused on low-cost and interaction they eventually started to have their own customers. Since then Kivnon Logistica's AGVs are present in several facilities around Europe including Jonson Control's, Seat's, Nissan's, Ford's, Volkswagen's and ABB's factories [28]. The AGVs are used for a variety of functionalities, where the most common one is to connect automatically several distant production lines.

2.2 AGV model

This project aims for a shopping-cart like AGV: an AGV that follows the user and has a space to deposit items. Nevertheless, as the mechanical design process of an AGV lasts long, Kivnon's K11 AGV will be used for testing purposes. The K11 is a unit load vehicle type AGV that drives under carts, hooks to them and pulls. It is able to move forward and backwards.

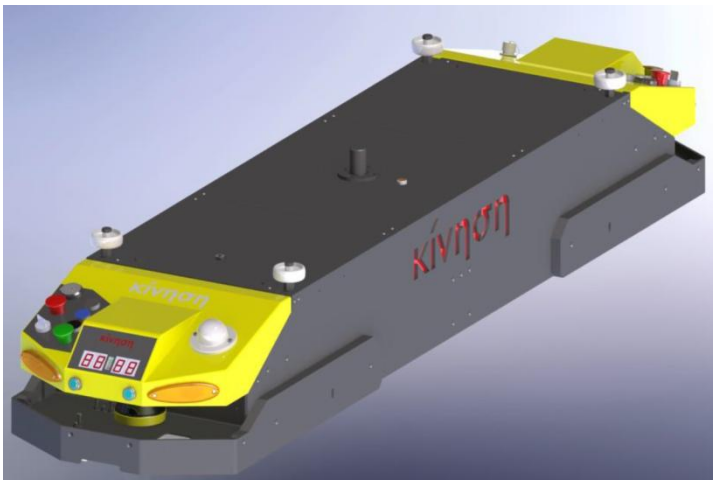


Figure 5: Kivnon's AGV K11 Mouse

2.3 States

The AGV can perform several actions, which all correspond to some states:

- On/Off
- Armed/Unarmed (the AGV starts unarmed, it has to be rearmed to work. If it stops because of an error it unarms itself for security reasons)
- Stopped/Moving forward/Moving backwards
- Pin up/ Pin down
- Alert/Warning
- Charging battery

2.4 Interaction with the environment

The AGV needs information about the surroundings to interact with machinery. This information is obtained through sensors or other devices. Thanks to this information the AGV is able to execute autonomously operations. According to Thomas B. Sheridan's and Raja Parasuraman's levels of automation the AGV would be fully autonomous, no human interaction is involved.

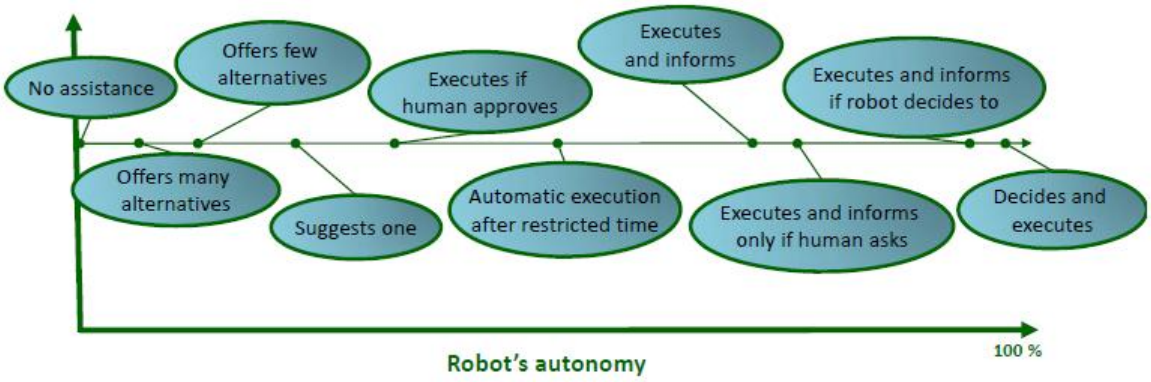


Figure 6: Thomas B. Sheridan's and Raja Parasuraman's levels of automation [29]

2.4.1 PLC

The PLC is the main core of the AGV. It serves as communication centre between all devices, automates them and computes the automatic control algorithms for navigation. The PLC is usually a Rockwell or Siemens PLC.

2.4.2 Magnetic sensor and magnetic band

Circuits, in which the AGV will navigate, are defined through a magnetic band. With the help of a magnetic sensor the AGV receives the information about its deviation to the centre of the band, enabling it to regulate the wheels speed in order to be on top of the band.

Figure 7: Magnetic sensor used in the AGV

2.4.3 RFID Tags

RFID Tags are small electronic circuits that can store some bytes of information. The AGV reads this information with an RFID antenna. The contained data can be localization information or orders. RFID Tags are placed on the floor.

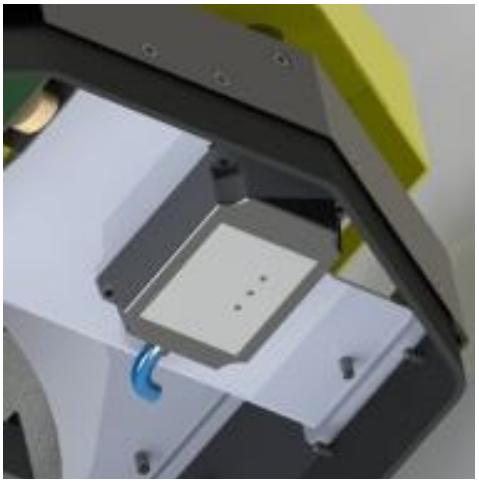


Figure 8: The RFID antenna scans the floor for RFID Tags.

2.4.4 Laser scanner

In order to not collide with obstacles the AGV is provided with a front a rear laser scanner that scans the surroundings. If an obstacle is detected it slows down velocity, if the obstacle is dangerously near it stops completely. The areas in which velocity is slowed down and stopped are called warning and alarm areas respectively and can be configured dynamically while the AGV is running.

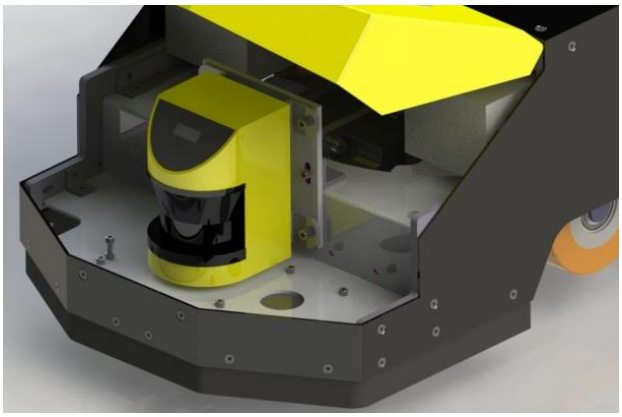


Figure 9: Laser scanner in the front of the AGV.

2.4.5 Pin

The AGV pulls from carts through a pin. When the AGV is under a cart the pin is raised or sunk to hook or unhook the cart.

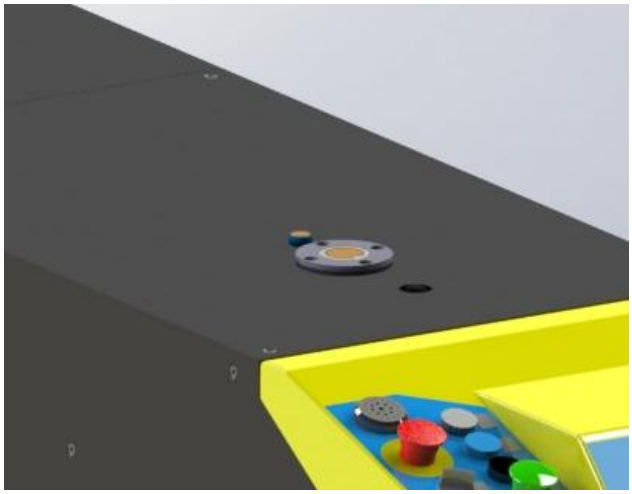


Figure 10: Pin to pull carts.

2.4.6 Traffic Light

A traffic light is a device that communicates through Zigbee protocol with AGVs approaching a crossing. It will only let pass 1 AGV at a time. Users can change the priorities of incoming branch through pushbuttons.

Figure 11: Traffic light. It has luminous pushbuttons to inform about the state of a crossing and change priorities. However it was conceived to work autonomously.

2.4.7 IO-Box

IO-Boxes are input-output arrays which communicate with AGVs. It enables them to exchange information and interact with machinery.

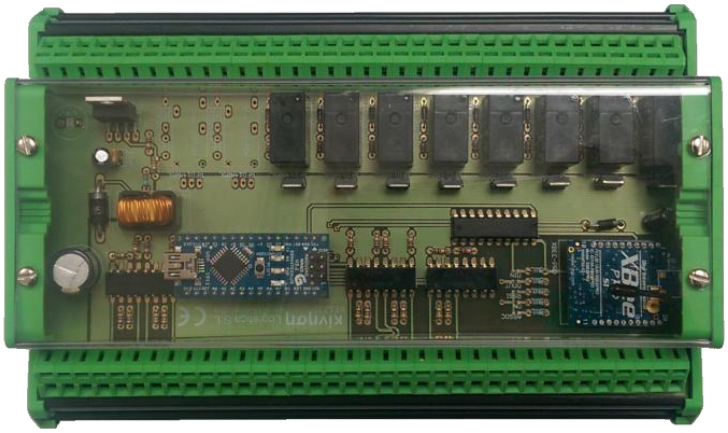


Figure 12: IO-Box. Through its input and output connectors it can interact with machinery.

2.5 Interaction with humans

The following three elements reduce the AGV's automation level to around 75% (Executes and informs).

2.5.1 Display

The display consists of 3 numeric indicators and a led bar. The first two digits will show the errors' or alerts' code number, the third will show the circuit's number. The led bar informs about the battery level.

Figure 13: AGV's display

2.5.2 Horn

The horn's purpose is only to inform acoustically the presence of the AGV.

2.5.3 AGV Monitoring Tool

The AGV Monitoring Tool is a software for PCs and tablets that enables exhaustive monitoring of AGVs. Position, velocity, circuits, orders and more can be viewed in a detailed manner through the graphical interface.

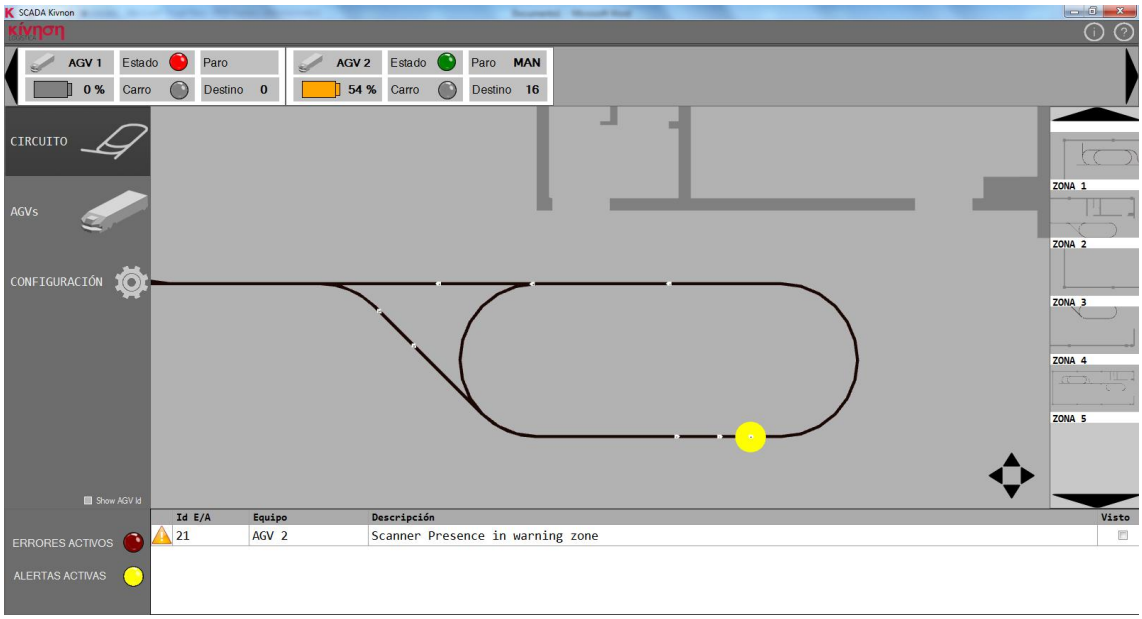


Figure 14: AGV Monitoring Tool main window

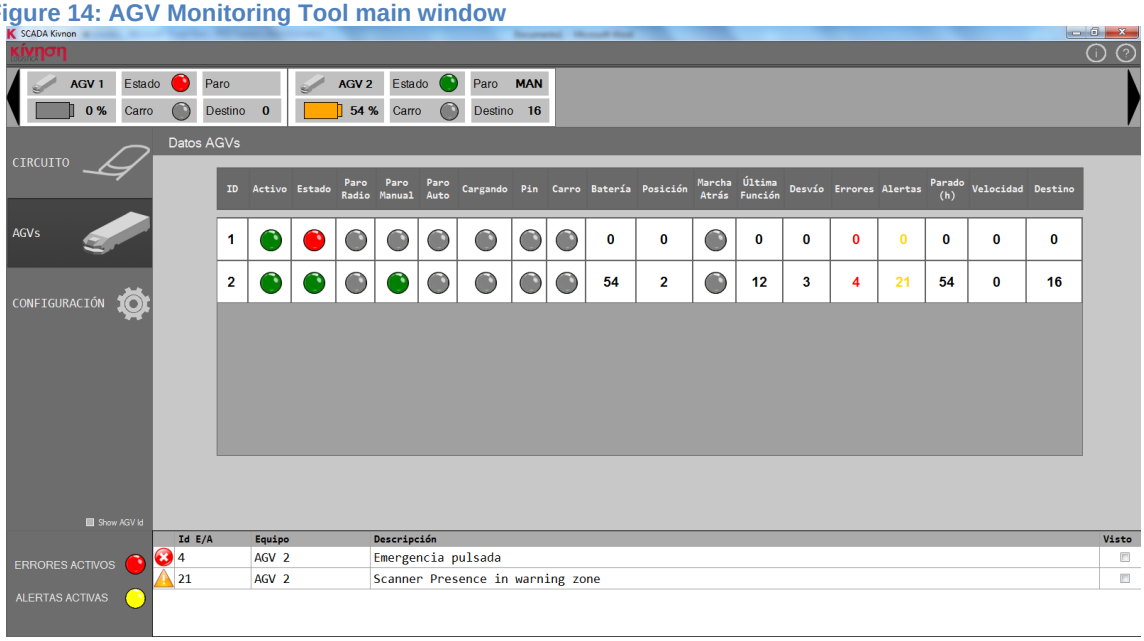


Figure 15: AGV Monitoring Tool AGV's properties windows

The next 4 elements reduce the AGV's automation level to around 15% (Offers few alternatives).

2.5.4 Control Panel

The control panel is the basic interaction of the AGV. It consists of 5 pushbuttons:

- Emergency stop
- Run/Stop button
- Change circuit
- Rise/sink pin
- Reset

It also includes an on-off switch to power on or power off the AGV.

Figure 16: Control panel

2.5.5 Stop&Go

Stop & Go is a device that communicates with the AGV through ZigBee protocol. It has the same function as the run/stop button, but in a remotely manner. It includes a status light that blinks if the AGV has been stopped by the device.

Figure 17: Stop & Go

2.5.6 Stop&Destination

Stop & Destination is a similar device to the Stop & Go. It enables to stop remotely an AGV and change its circuit's number.

Figure 18: Stop & Destination. It also exists as a software version for PCs and tablets.

2.5.7 Destination Supervisor

The destination supervisor software communicates through ZigBee protocol with a queue of AGVs which are stopped on a line. It also communicates with other devices called “Ordering Boxes”. Ordering Boxes are usually places far away in a production line. If the operator needs the presence of an AGV to take some material away or bring him material he will press the ordering box’s pushbutton informing the destination supervisor. The destination supervisor will then manage the queue of AGVs to send each one where it is needed.

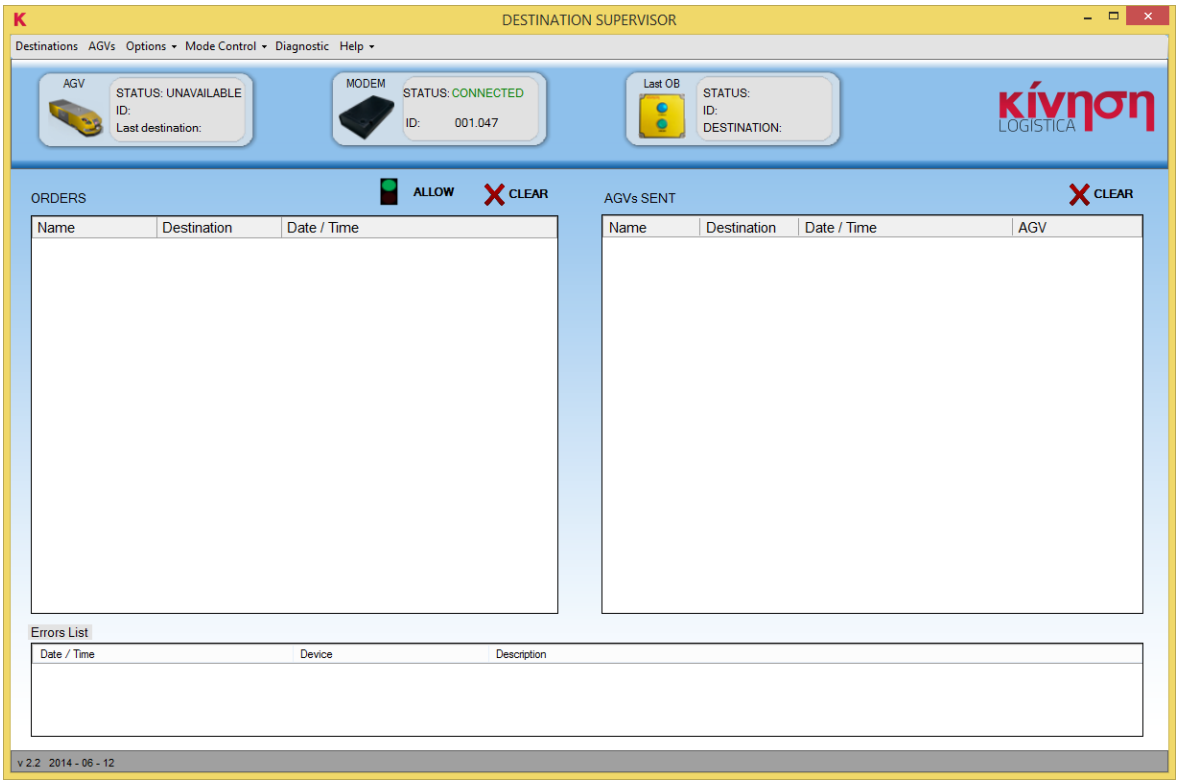


Figure 19: Destination Supervisor main window

To sum up, a PLC serves as core of the system, connected to a series of devices electrically or through some type of communication such as ZigBee or Ethernet. Those who interact with the environment and other machinery or AGVs rise the automation level. Thos who interact with humans drop the automation level.

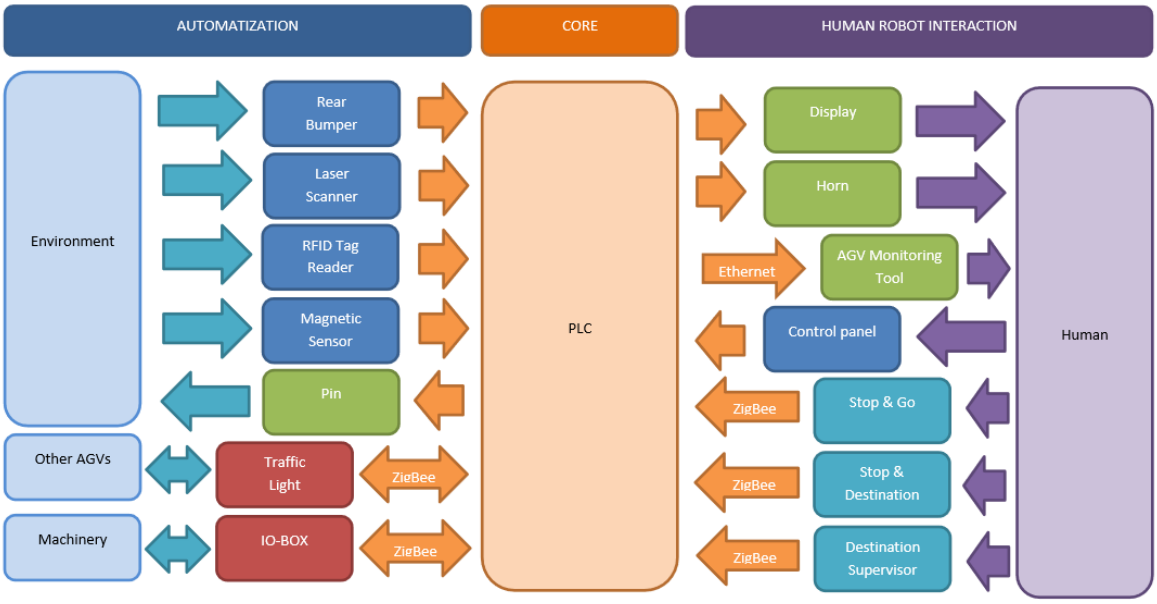


Figure 20: Scheme of information and interaction flow in the AGV

CHAPTER 3: DEVELOPMENT PLATFORM

In the investigation and experimentation phase of this project many different platforms where tested to choose the best one. Platforms include the controller (CPU), the image and depth sensor, the operating system in which the final product will run, the programming language and the programming environment.

3.1 Controller

The controller must be small enough to fit in the AGV and be able to connect to the Kinect's sensor's data stream, thus a system with USB connection and an operating system is needed. The first solution was to use an ODROID-XU3, a 2.0 Ghz octo-core credit card sized PC. The ODROID-XU3 is capable of running Ubuntu 14.04 or Android 4.4. ROS is easy to install on it with Ubuntu. Several developers have published projects involving ODROID-XU3 and Kinect sensor so it seemed the right solution.

Figure 21: ODROID-XU3 credit sized PC

Nevertheless the Ubuntu version capable of being installed on this system is the ARM variant. ROS for Ubuntu ARM is more limited than the desktop version. The ROS key node for this project "skeleton tracker" is not available for Ubuntu ARM. Also de "Nite" middleware library is not available for Ubuntu ARM. This is probably the mayor reason, as the "skeleton tracker" node is dependent of "Nite" and "Nite" is not open-source.

Thus a system that works with desktop operating systems was needed. The solution is a Intel NUC. Intel NUCs are barebone type PCs. With a size of 117 x 112 x 35mm it is small enough to fit in the AGV. Its i5 4250U processor enables it to run Windows 8 or Ubuntu 14.04 Desktop with ease. It also has a connection header to control power signal, which is very convenient to turn it off safely once the AGV turns off.

Figure 22: Intel NUC D54250WYK

3.2 Sensors

The main key of this project is the depth sensor. Since 2010 several models have been launched. The original Microsoft Kinect for windows sensor was the first choice. It features an RGB camera with a 1280x960 resolution and a depth camera. The technology used by the depth camera was developed by PrimeSense. It works emitting an infrared light pattern which an infrared camera captures. The sensor can reconstruct a 3D scene interpreting the deformation of the pattern caused by the objects of the environment. The minimum and maximum depths that the Kinect recognizes are 0.8 meters and 4 meters respectively. Once the depth image is obtained the GPU of a PC is able extract skeletons through a random decision forest algorithm [14].

Figure 23: Microsoft's Kinect for XBOX 360

Unfortunately at December 2013 the company Apple bought PrimeSense forcing Microsoft to stop producing the Kinect sensor. If Kivnon Logísitica is successful with this product it cannot depend on a discontinued product.

The direct alternatives are Asus' Xtion sensors. These sensors a very similar to the Kinect and work with the same principle. They are smaller, can detect depth of 0.4 to 4 meters and do not need a power supply as they take the energy through the USB port. But they also use PrimeSense's "Nite" middleware to detect skeletons, thus are not a good solution.

Figure 24: Asus' Xtion Pro Sensor

With the launch of Microsoft's home-console "XBox One" a new generation of depth sensors came out. First known as "Kinect for Windows v2" and now as "Kinect for XBox One" it presents a much higher depth precision than its predecessor. Instead of using the infrared-light pattern technique it uses time of flight technology making it not only more precise and stable, but also robust to sunlight (the sun emits large amount of infrared light, thus "erasing" the emitted pattern of the old sensors). This sensor provides also a 1920x1080 RGB image, infrared sensor and a 8 array microphone. This amount of dataflow needs a USB 3.0 to work properly. 2 USB ports of the Intel NUC are 3.0-compatible.

Figure 25: Microsoft's Kinect for XBOX One

3.3 Operating system

The NUC can either work with Windows or Ubuntu. Windows 8.1 has been decided as Microsoft provides a powerful SDK to develop Kinect applications on the Windows platform.

The university of Bremen released a driver to work with Kinect for XBox One on ROS [3] but the compatibility is still low.

3.4 Programming language and environment

Kinect for XBox One's SDK gives the possibility to work in C++, C# and Visual Basic. As C# is powerful and simple it has been chosen as the language to develop in. The first idea was to develop a Windows RT App (Windows Store App) due to its simple and minimalistic interface and due the fact that Windows 8.1 only enables Kiosk mode for Windows RT Apps. The Kiosk mode is a mode in which the operating system launches directly an application and locks the screen to it.

Windows RT Apps have some limitations. In Windows 8.0 this apps could not access to serial ports and in Windows 8.1 they need a CDC ACM compliant USB serial port converter. Such devices have been poorly deployed and it is difficult to find any on the market. The serial port is needed to communicate the system with the PLC of the AGV. An alternative would have been using the Ethernet connection, but this would require the use of an OPC which does not exist for Windows RT apps or, alternatively, use sockets, for which the closed Rockwell's PLC Ethernet frame's structure would be needed.

Thus the author of this project has decided to code a WPF (Windows Presentation Foundation) application, which have a similar interface philosophy to Windows RT apps. WPF applications are compatible with any serial device.

As programming environment Visual Studio 2013 has been decided to be used as it is extremely powerful for Windows' applications coding and makes it easy to add new packages to an existing project.

CHAPTER 4: DATA PRE-PROCESS

The Kinect sensor is capable of tracking up to 6 bodies and provides the position of each of the body's joints. This raw data is useful for representation (for example 2D image representation) but not accurate or stable enough for gesture recognition.

4.1 Filtering

First of all data can be shaky, thus filtering is necessary [67].

4.1.1 Average filter

The most basic and easy to implement filter is the average filter. It responds to the next formula:

$$filtered\_data_{f_c} = \frac{\sum_{f=f_c-n+1}^{f_c} raw\_data_f}{n} \quad (1)$$

Where f_c is the current frame, n is the number of frames used by the filter and raw_data_f is the raw data of the frame f_c.

To obtain a smooth enough signal to be useful n has to be large. For example if a single frame gives the position of a joint with an error of 10 cm and at least 2 cm precision is needed, 10 frames will be necessary to archive this precision. This means that the data of the whole 10 frames must be stored. In addition mean filter provoke a huge delay of n/2 frames.

4.1.2 Kalman filter

The drawbacks of the average filter make it not suitable for the purpose of gesture recognition. A faster and memory lighter filter is needed. The Kalman filter, also known as Kalman estimator, is an algorithm that produces an optimal estimate in the least square sense of the actual value of a state vector from noisy observations. [37]

To use the Kalman filter a dynamic model of the type $x_k = F*x_{k-1} + G*u_k$ and $y_k=H*x_k$ for each joint of the human's body is needed. As this model depends on the will of the human (defining the human model is highly complicated) the simplified model $x_k = x_{k-1}$ and $y_k = x_k$ will be used.

The algorithm for this particular model type takes the form:

A priori prediction:

$$\hat{x}_{k|k-1} = \hat{x}_{k-1|k-1} \quad (2)$$

$$P_{k|k-1} = P_{k-1|k-1} + Q_k \quad (3)$$

Update step:

$$\tilde{y}_k = z_k - x_{k|k-1} \quad (4)$$

$$S_k = P_{k|k-1} + R_k \quad (5)$$

$$K_k = \frac{P_{k|k-1}}{S_k} \quad (6)$$

A posteriori prediction:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k \quad (7)$$

$$P_{k|k} = (1 - K_k) P_{k|k-1} \quad (8)$$

Where $\hat{x}_{k-1|k-1}$ is the state estimation of the last iteration, $\hat{x}_{k|k-1}$ is the a priori state estimation, $\hat{x}_{k|k}$ is the a posteriori state estimation, $P_{k-1|k-1}$ is the estimate covariance of the last iteration, $P_{k|k-1}$ is the a priori estimate covariance, $P_{k|k}$ is the a posteriori estimate covariance, Q_k is the estimated error in the process, R_k is the estimated error in the measurements, z_k is the current sensor value, $\tilde{y}_k$ is the innovation, S_k is the innovation covariance and K_k is the optimal Kalman gain.

R_k and Q_k maintain its value through time and are not updated. R_k can be obtained as the covariance of the measurements. Q_k can be adjusted experimentally. As the model attempts that the estimation remains equal to the last estimation a high Q_k is needed in order that the estimation converges fast to the actual value.

$P_{k|k}$ changes over time so $P_{0|0}$ must be set as a parameter. As the initial condition are not known $P_{0|0}$ must be large.

The Kalman filter only needs to store the last iteration estimations and with a high Q_k it is fast, so this filter is suitable for the application.

As the Kalman filter presents several advantages over the mean filter it will be the Kalman filter which is implemented in the project.

4.2 Data transformation

The data of the joints is given in absolute position in a fixed world frame. The reference frames of this data can be transformed or the data can be even converted into angles. This section discusses the advantages and drawbacks of each transformation.

4.2.1 Absolute position

Using the raw data has the big advantage that no calculation is needed. Nevertheless it cannot be used as the data would be dependent on the user's position, thus a gesture performed in position x would be different to a gesture performed in position y.

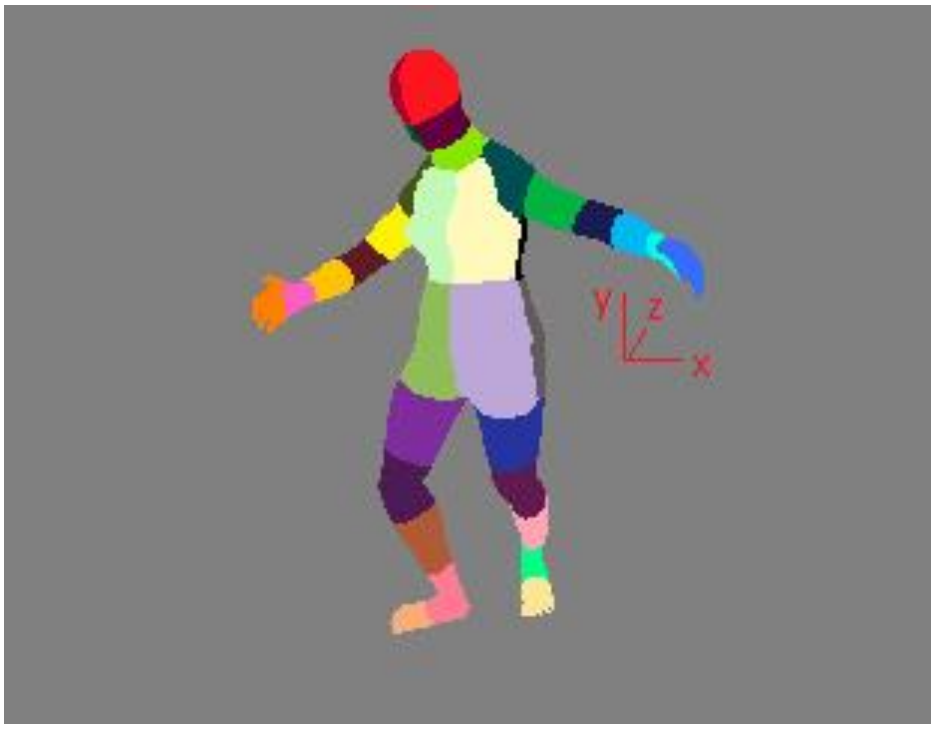


Figure 26: User's body respect to the world frame

4.2.2 Relative position

First all space points of the joints are rotated in such a manner that the user is facing to the camera. Then the reference frame is set to one of the joints of the human body, for example the middle spine.

Figure 27: Reference frame is fixed to user's body

Thanks to this rotations and translations the joints of the body do not depend on the position neither on the orientation of the user.

The main disadvantage of using position in general is that it depends on the user's body complexion, gestures performed with a long arm will be different than those performed with a shorter arm. To avoid this Sait Celebiet. al. [7] proposes to scale the body in function of the user's height although every human has slightly different body relations.

In addition the distance error of a pose compared to a model propagates through each joint and distance to the reference frame. For example if a user is holding his arm straight forward and other user is doing the same gesture but with one degree deviation in each joint the distance of the hands could be big enough to not being detected as the same gesture.

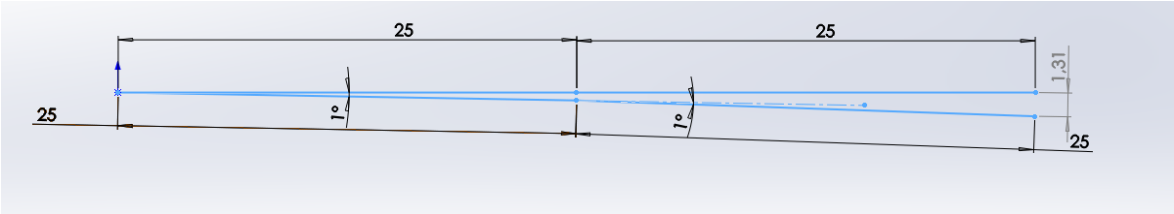


Figure 28: Example to show how error is propagated through distance

4.2.3 Absolute angle

As seen using space position of joints is not an option for the purpose of gesture recognition. Using angles instead solves the problems of error propagation and the user's body complexion. Absolute angles can be obtained starting from the vector that 2 joints form (link). However absolute angles depend on the users pose and are therefore not suitable.

4.2.4 Relative angle to body

Obtaining the angles of each link in reference to the body eliminates this dependence. Nevertheless a drawback that is present on both angle type is that for example if a gesture of person A has the forearm parallel to the forearm of person B the angles will be the same even if the upper arms are in different poses, causing a possible match between two different gestures. Roll angles are also difficult to obtain.

4.2.5 Relative angle to parent

To avoid the last mentioned drawbacks each links' angle should be taken in the reference frame of the parent link. This means that for example the forearms angles depend on the upper arm reference frame. Thus for each link 3 joints are needed. In the case of the forearm the shoulder and elbow joints are needed to compute the upper hand reference frame and the hand (and elbow) joint is needed to compute the angles.

In addition if roll is wished to be obtained 2 more joints are needed. These are the starting joints of the parent of the parent link and the ending joint of the child link. With the first one a 0 reference for the roll is obtained and with the second one the actual torsion of the link.

For simplicity Tait-Bryan angles will be used. These present singular points which should be avoided. Luckily using the relative angles to parent these singular points appear in very unnatural position, so they will not disturb the gesture recognizer algorithm.

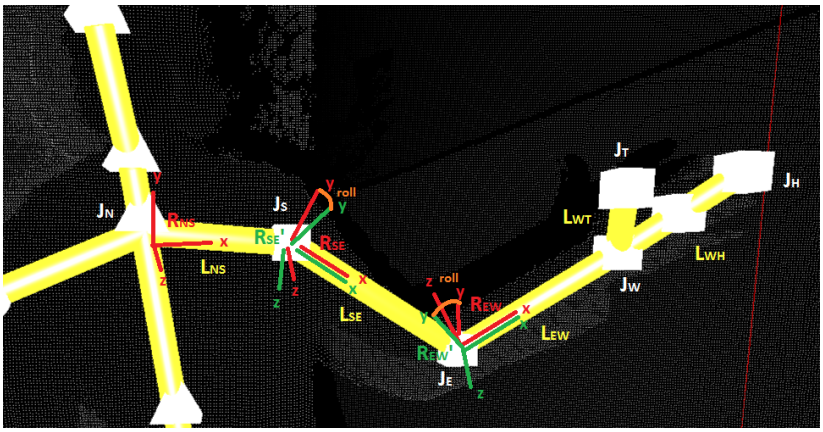


Figure 29: Reference frames for each link

The calculations donde to obtain the angles can be found in the annex.

CHAPTER 5: GESTURE RECOGNITION ALGORITHMS

The aim of gesture recognition is to identify and classify gestures a user performs. Gesture can include whole body motion, hand motion, face gestures and many more. This project focuses on gestures made with the whole right arm. In the future both arms will be implemented. Once a gesture is recognized and classified it send a command to the AGV to perform a functionality, enabling a natural communication between human and AGV.

Gesture recognition algorithms for body motion can be divided in 3 big families [49]: 3D model-based algorithms, skeletal-based algorithms and appearance-based models. Skeleton-based algorithms use stereo vision to obtain a skeleton model of the user which is then compared to a database to identify gestures. 3D model-based algorithms can be thought as a more complex version of skeleton-based algorithms in which much more data is involved. Appearance-based algorithm consists in matching statistical models of object shapes and appearance to a 2D image. They are much more complex algorithms and are used mostly to identify face gestures and not body gestures.

As the Kinect sensor already provides the skeleton data the most convenient algorithm family is the skeletal-based one. There exist mainly 3 skeleton-based algorithms: Hidden Markov Model [50], whose performance is often not so good, Neural Networks with Hidden Markov Models, which were conceived as improvement of the first mentioned algorithms [4], and finally Dynamic Time Warping algorithms [51], which measure the similarity between two signals that may vary in speed.

The skeleton joints that will be analyzed are:

- Neck
- Shoulder centre
- Shoulder right
- Elbow right
- Wrist right
- Hand tip right
- Thumb right

DTW is an efficient and easy-to-understand algorithm; therefore it was chosen to be used in this project.

5.1 Basic DTW

The DTW algorithm is meant to compare two equal shaped signals that vary in speed. The classic way of comparing two signals is to sum the Euclidean distance at each time

sample. If both signals have the same shape but differ in speed the obtained sum will not be 0.

DTW compares both signal synchronizing them adding and deleting time samples of each signal. It works in a similar way to the approximate string matching algorithm used in spelling checker.

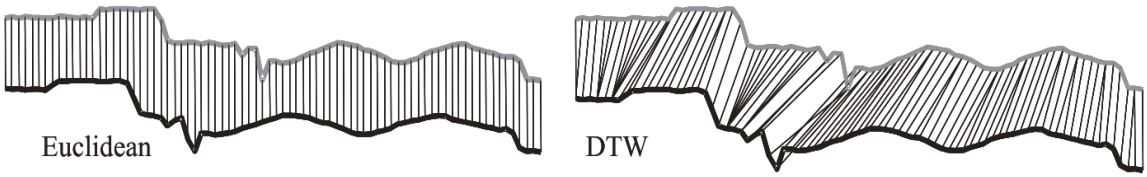


Figure 30: Euclidean distance and DTW distance [6]

Suppose two different signals, signal A has x_t time samples and signal B y_t time samples. The accumulated distance to the pair sample (x,y) is the Euclidean distance of sample x compared to sample y plus the minimum of accumulated distance of pairs (x-1,y-1), (x,y-1) or (x-1,y). If the minimum is the pair (x-1,y-1) both signal have the same speed. If the minimum pair was (x-1,y) then signal A is slower around this sample than signal B. Finally if the pair(x,y-1) was the minimum signal A is faster than signal B.

To obtain the accumulated distance for pair (x_t,y_t) the rest of accumulated distances must be obtained first. A matrix can be constructed in which each cell is a time sample comparison.

```
Input:

sA[xt]

sB[yt]



DTWMatrix[xt+1][yt+1]

for x=0:xt

        for y=0:yt

                DTWMatrix[x][y] = oo

DTWMatrix[0][0] = 0



for x=1:xt

        for y=1:yt

                cost = |sA[x]-sB[y]|

                DTWMatrix[x][y] = cost + min(DTWMatrix[x-1][y-1],

                                             DTWMatrix[x][y-1],

                                             DTWMatrix[x-1][y])

Return DTWMatrix[xt][yt]
```

Algorithm 1: Basic DTW

For some applications it may be of interest to know the synchronization shape. This can be done drawing for each cell its origin cell and once all are drawn search for the optimal path.

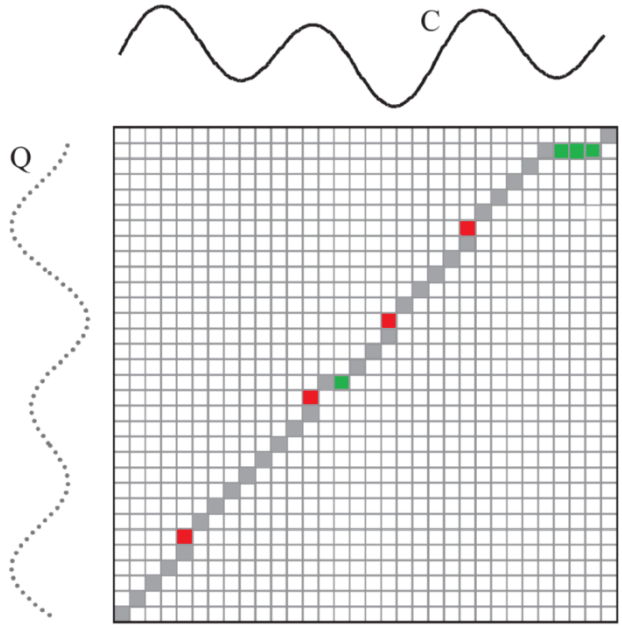


Figure 31: Synchronization of Q respect to C. Red squares mean Q is being slower than C, green means Q is being faster than C [6].

5.2 DTW improvements:

5.2.1 Multi-Dimensional DTW

The previously mentioned algorithm works fine for one-dimensional signals, but most applications (including the one of this project) are multi-dimensional. G.A. ten Holt et. al. presented an algorithm to include all dimensions and discriminate the ones that have no information [5]. First all dimension are normalized to zero mean and unit variance. Next, while calculating the DTW matrix the cost will be any p-norm, for example the 1 norm: the sum of the absolute differences in all dimensions f (features).

$$DTWmatrix_{cost}(x,y) = \sum_{f=1}^{F} \left| S_{f,x}^{a} - S_{f,y}^{b} \right| \quad (9)$$

Gestures in which there exists at least 1 dimension that is static are problematic while normalizing the variance. Normalized static dimension will contain only noise. This is where dimension selection comes in. Dimension selection consists in eliminating those dimensions that have a variance below a threshold. The dimensions to be erased have to be selected off-line for each gesture. Instead of discarding the dimension all dimensions can be weighted. This will be discussed in the Weighted DTW section.

5.2.2 Mean and variance discrimination

Normalizing dimension supposes a huge loss of information. The next 2D example will make this issue evident.

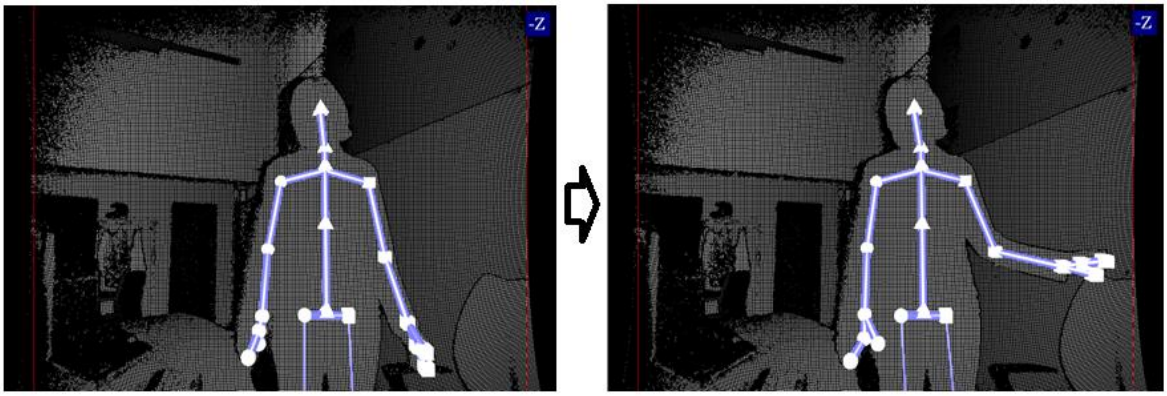


Figure 32: Gesture A

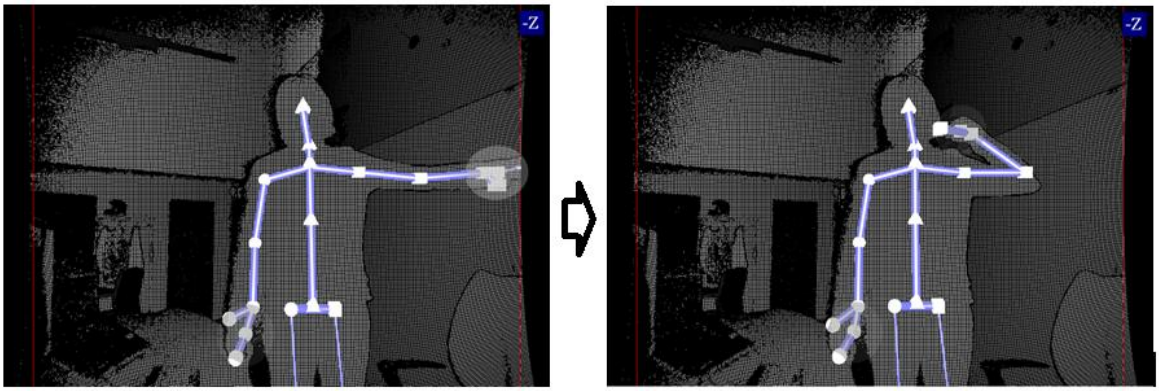


Figure 33: Gesture B

It is obvious for a human perception that Gesture A is not the same as Gesture B. As signals they also look at first instance as different gestures.

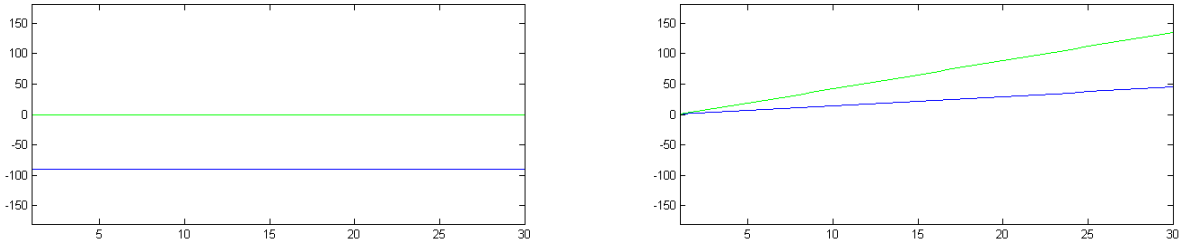


Figure 34: Blue for gesture A, green for gesture B. Left for upper arm angle, right for forearm angle.

But once both gestures are normalized they look exactly the same.

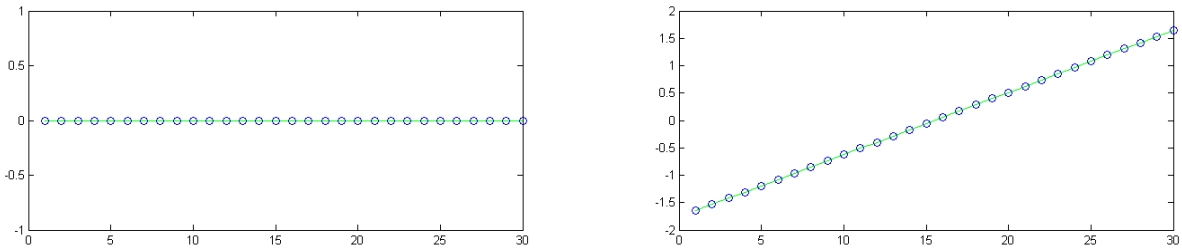


Figure 35: Figure 13's normalized gestures. Upper arm angles have not been normalized to unit variance as their variance was 0.

To avoid this loss of information this project introduces a novel technique that discriminates gestures by mean and variance. Gestures are normalized as always but the original mean and variance is stored as a parameter of each signal. Before applying the DTW algorithm the system checks if the gesture to compare has a similar mean and variance to those of the database. Thanks to this technique information is not lost and moreover the identification process can be speeded up as most models can be discarded with only 1 or 2 comparisons.

5.2.3 Weighted DTW

To improve the obtained results Sait Celebiet. al. [7] introduced the weighted Dynamic Time Warp. It could be thought as a more sophisticated multi-dimensional DTW with weights obtained by a machine learning algorithm.

The main principle of this technique is to find weights that increase discrimination between gestures of different types and reduces it in gesture of the same type by searching a discrimination ratio similar to Fisher's Discrimination Ratio [8].

The formulae applied in this project can be found in the annex.

5.2.4 Derivative DTW

DTW algorithm can produce pathological results. Sometimes the algorithm may explain variability of the value-axis warping the time-axis, resulting in unintuitive alignments. Concerned about this problem Eamonn J. Keogh and Michael J. Pazzani proposed the DDTW algorithm (Derivative DTW) [36]. The DDTW changes the original features for its local derivatives. By doing so they have demonstrated to obtain improved synchronizations. However the obtained error-distance is highly depended on velocity, feature which optimally should be neglected in gesture recognition. Therefore the use of DDTW was rejected for this project.

5.3 Time optimization:

In gesture recognition it is important to maintain a constant fps rate; it is important that information flow keeps the same. Even though the DTW algorithm is robust against speed variations it is not completely robust against loss of information.

DTW is however a very computational expensive algorithm. If a database has 10 different gestures, each one with 100 model samples and each model has 30 time samples, a new gesture should be compared to 1.000 models using the DTW algorithm. Each time the DTW algorithm is executed is must perform around 18.000 subtractions, 6.400 additions, 5.400 multiplications, 4.500 comparisons, 3.600 moves and 900 square roots operations. This means that for each gesture identification 390 million operation must be done. If a constant fps rate of 30 is wanted to be reached 1,7 GFLOPS are needed, computing capacity that is reached for example by a cluster of more than 60 Raspberry Pi [9]. It becomes clear that time optimizations techniques must be applied to make the algorithm functional.

5.3.1 Sakoe-Chiba band

Sakoe and Chiba were working on speech recognition with dynamic programming algorithms (DTW is also a dynamic programming algorithm) and introduced a maximum windows size while looping around columns of the matrix [10]. The windows size cannot be smaller than the difference of lengths of both signals to be compared. The algorithm changes to:

```
Input:

sA[xt]

sB[yt]

w = max(w, abs(xt-yt))

DTWMatrix[xt+1][yt+1]

for x=0:xt

        for y=0:yt

                DTWMatrix[x][y] = oo

DTWMatrix[0][0] = 0

for x=1:xt

        for y=max(1,x-w):max(m,x+w)

                cost = |sA[x]-sB[y]|

                DTWMatrix[x][y] = cost + min(DTWMatrix[x-1][y-1], DTWMatrix[x][y-1],

                                             DTWMatrix[x-1][y])

 Return DTWMatrix[xt][yt]
```

Algorithm 2: Sakoe-Chiba band or windows DTW

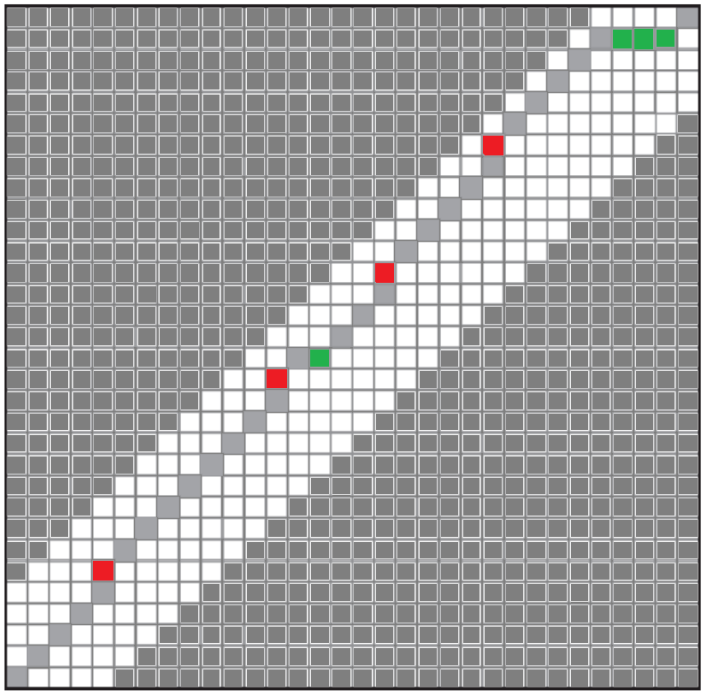


Figure 36: Sakoe-Chiba band example with a windows size of 6. Dark grey cells will not be computed.

Applying the Sakoe-Chiba band reduces the computation time drastically. For example a 30x30 matrix with a Sakoe-Chiba band with a windows size of 5 will loop only around 250 cells instead of 900.

The use of this band does not only speed up the identification process but also introduces local constraints [11]. This means that even though 2 signals could be synchronized with a 0 distance by slowing down one signal heavily and then speeding it up again the resulting distance will not be 0 as the optimal path can only be inside the band. In conclusion: 2 signals that vary too much in speed will not be identified as the same.

5.3.2 Itakura parallelogram

Fumitada Itakura [21] proposed a band similar to those proposed by Sakoe and Chiba. Instead of being a constant width band it is a parallelogram. The Itakura parallelogram has a similar time reduction ratio as Sakoe-Chiba band but different local constraints. It will permit large speed ups and slow-downs at the middle of the signal, but will limit it at the start and ending.

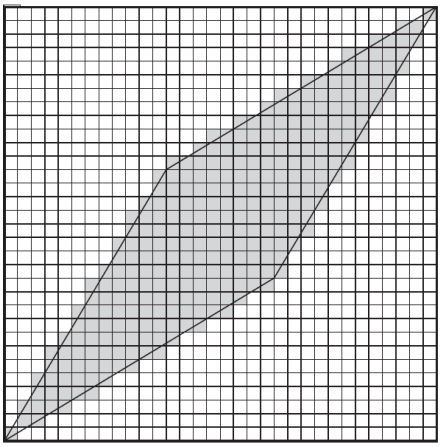


Figure 37: The Itakura parallelogram

The main drawback of the parallelogram is that it forces both signals to synchronize at the last samples. If a gesture is faster than other gesture it will reduce its distance score. To avoid this, gestures to be compared are divided in several signals of different lengths. However this slows down the process heavily.

5.3.3 Triangle band

In order to avoid the Itakura parallelogram's drawbacks this project introduces a novel band type. Based on the parallelogram, the permitted change of speed increases over time, but in this case it does not shrink down after the middle point.

The last line of the DTW algorithm is also changed to return not the (x_T,y_T) cell, but the minimum of $(x_T,y_T:y_{T-w})$ where w is an arbitrary windows size. This allows the gesture to be classified even if it is faster than the model with a 0 DTW distance. If the gesture is slower it will not be able to be classified because there would exist a lack of information of the last part of the gesture.

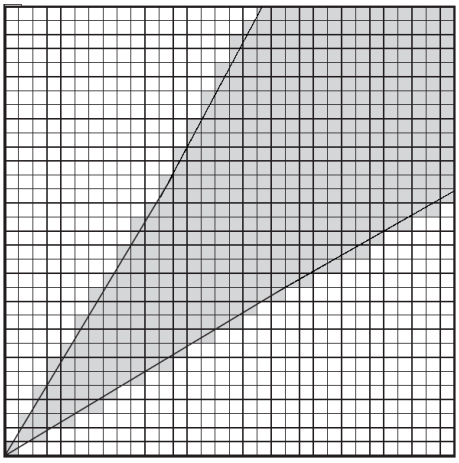


Figure 38: Proposed triangle band

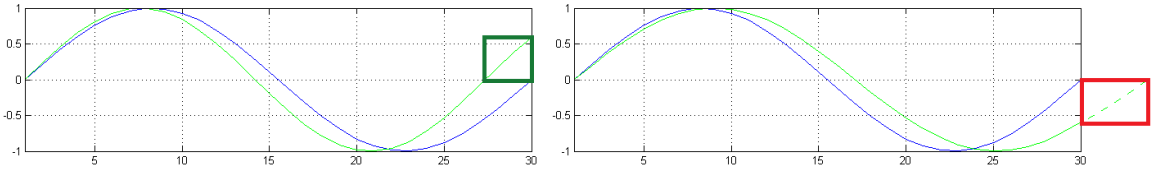


Figure 39: Blue signal is the model gesture. Green signal is left an accelerated gesture, right a decelerated gesture. The dark green box shows information that will be ignored as the gesture has finished earlier, thus it can be identified. The red box shows information that is missing to complete the gesture. Because of the lack of information the gesture will not be identified.

5.3.4 Keogh bound

The techniques discussed till now are local constraints techniques. An even higher speed up can be obtained introducing lower bounding. Lower bounding techniques are based on calculating a lower bound for a sequence (which is fast to obtain) and only calculating the DTW if the lower bound is smaller than the lower bound so far.

The basic algorithm is [6]:

```
best_so_far = infinity

for all sequences in database

        LB_dist = lower_bound_distance(sA,sB)

        ifLB_dist<best_so_far

                true_dist = DTW(sA,sB)

                if true_dist<best_so_far

                        best_so_far = true_dist

Return best_so_far
```

Algorithm 3: Low Bounding Sequential Scan

There exist several ways of calculating a lower bound distance. Eamonn Keogh suggested one of the most efficient ones. Keogh lower bound must work with a constraint technique. Each sequence will define two new sequences U and L:

$$U_f = \max(sB_{f-r} : sB_{f+r}) \quad (10)$$

$$L_f = \min(sB_{f-r} : sB_{f+r}) \quad (11)$$

Where X_y is the value of sequence X in the frame y and r is the windows size of the local constraint. For Sakoe-Chiba r is constant, for Itakura parallelogram and Triangle bound it depends on f.

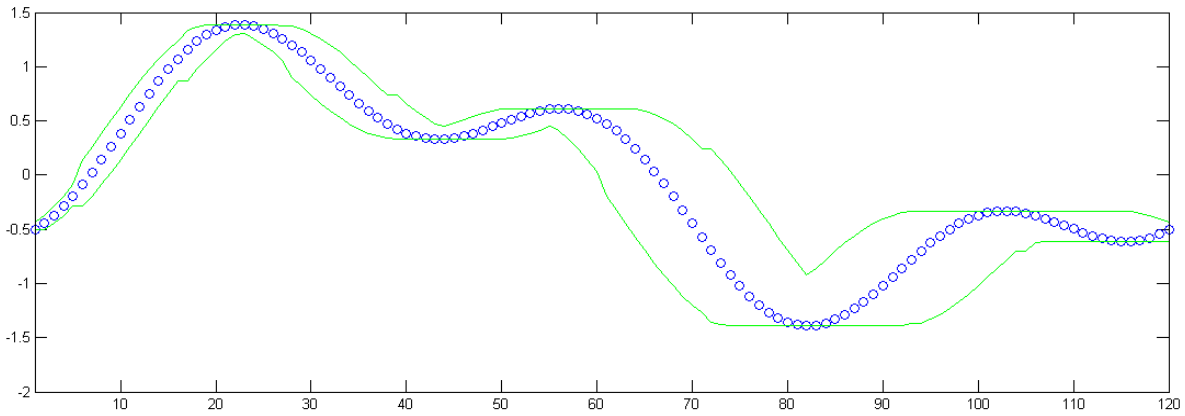


Figure 40: Blue dots are S_B sequence, green lines are U and L sequences obtained from a Triangle band with r=1+frames/12.

The lower bounding measure for the DTW is then:

$$LB_{Keogh(sA,sB)} = \sqrt{\sum_{f=1}^{t} \begin{cases} (sA_f - U_f)^2 & if sA_f > U_f \\ (sA_f - L_f)^2 & if sA_f < L_f \\ 0 & otherwise \end{cases}} \quad (12)$$

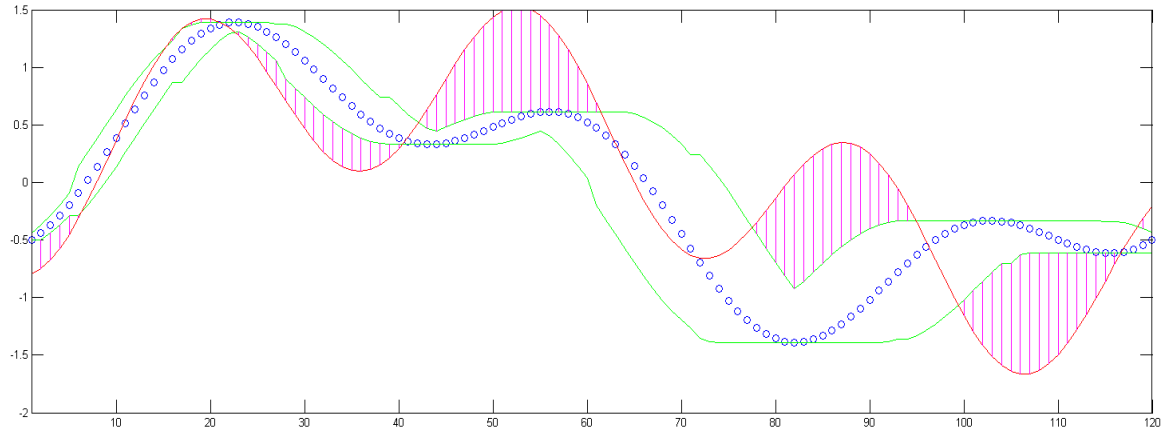


Figure 41: Red line is S_A sequence. LB_Keogh is the second norm sum of all magenta lines.

There exist even better lower bounds. Daniel Lemire proposed the LB_improved based on LB_Keogh and demonstrated that it is slightly more efficient [13].

All above presented methods have been tested in this project in the experimental phase. The best results were obtained using the mean and variance discrimination together with the weighted DTW, triangle band and Keogh bound, thus those have been implemented in the final product.

CHAPTER 6: USER RECOGNITION

Users must be sometime recognized for the sake a functional product. For example when the tracking of the user is interrupted for any reason (for example he has left the field of vision of the Kinect sensor), besides of ending Human-Robot communication, the user must be recognized again to restart the tracking.

The following technique was thought for the Kinect for Xbox 360 which has a poor colour camera resolution. This technique consists in identifying the colour of the t-shirt the user is wearing. A square of colour pixels is analyzed from the users t-shirts. This square grows and diminishes its area in function of the distance of the user to the AGV, in such way that the square is always smaller than the t-shirt. The mean colour is computed, transformed and compared to the database to recognize de user.

Figure 42: Square from which colour pixels are obtained

With the Kinect for Xbox One face recognition could be applied thanks to the 1920x1080 colour images obtained. However the t-shirt colour identifier works well so this project will not apply face recognition.

6.1 Colour

Colour can be difficult to handle. Light's intensity and hue have a big influence on the sensors of cameras. In computer science colour is usually represented as an array of 3 values. There exist different representation in which each cell of the array has a different meaning.

6.1.1 RGB

RGB stands for red, green and blue and is the most usual representation. At first glance it can seem intuitive but it is in fact a very non-linear representation. For example if a pixel is very colourful and in the next instance less colourful, some of the 3 properties will rise and other will sink. If light brightens up all properties rise, but with different velocities. This means light affects all 3 properties. The AGV will usually move around a factory, where

light intensity changes constantly. Therefore a colour representation where at least one property is independent of light is desirable.

Figure 43: RGB colour representation

6.1.2 HSL and HSV

HSL stand for Hue, Saturation and Light and HSV for Hue, Saturation and Value (or also brightness). Both are cylindrical-coordinate representations of colour inspired by the colour wheel. They are transformed from RGB with the intention to be more intuitive. Both use the term “Saturation” but their mathematical definitions are not the same.

Both representations can be imagined as a cylindrical tower. The angle around the vertical axis corresponds to the hue. Hue is defined technically in the CIECAM02 model as “the degree to which a stimulus can be described as similar to or different from stimuli that are described as red, green, blue, and yellow". [41] The primary colours red, green and blue are located with a 120º interval: 0º for red, 120º for green and 240º for blue.

The distance to the centre of the cylinder represents the saturation. In other words the colourfulness. Colours in the centre are achromatic.

Finally the height deals with lightness or brightness depending on the model. A colour with few lightness or brightness is dark. Bright colours present high lightness or brightness.

Both representation present singular zones: For the colour black (lightness or value equal 0) hue and saturation are undefined. For achromatic colours (0 saturation) hue is undefined. In HSL if lightness is maximal (colour white) hue and saturation are undefined as well.

As HSL present an addition singular zone, the project will use HSV colour representation.

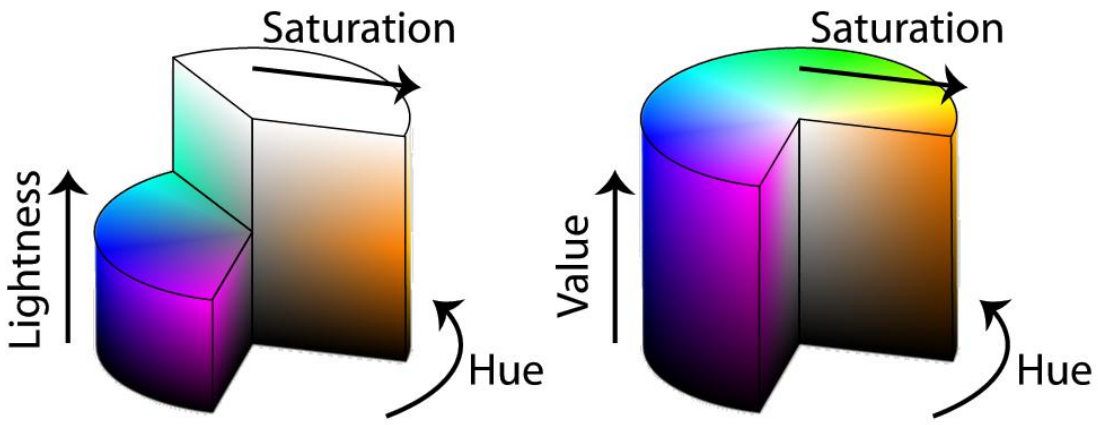


Figure 44: HSL (left) and HSV (right) colour representation [15]

The main property to be compared is hue as it should not be affected by light conditions. If hue differs an arbitrary value colours will not be considered as the same. However in dark coloured t-shirt or achromatic coloured t-shirt hue will be ignored, as it is not well defined. Instead saturation and value will be compared.

In order to increase the effectiveness of the user recognition, users will be asked to wear specific colour t-shirt, for example yellow or green.

CHAPTER 7: AGV NAVIGATION

The whole system would not be functional if the AGV does not move. Two different systems could be implemented for the AGV's navigation. Currently the AGV navigates on top of a magnetic band that defines circuits around the plant.

In the first proposed system the AGV would still navigate on top of these defined circuits and would only get velocity orders depending on the users distance, therefore limiting drastically the AGV's degrees of freedom. However it would enable an easy implementation in all Kivnon Logistica's facilities, as once human-robot interaction is terminated the AGV would not need to search again for the magnetic tape to continue its navigation fully autonomous. This navigation system will be called magnetic tape navigation.

The second proposed system is to let the AGV move freely around the plant once human-robot interaction is started. The AGV would follow the user wherever he goes. However once the human-robot communication is terminated the AGV would not be able to move as it would have abandoned the magnetic tape not being able to localize itself. To make this system profitable another main navigation system should be implanted like for example SLAM through scanner, beacon navigation, RFID tag navigation or satellite navigation. This navigation system will be called free navigation.

Both navigation systems will be described, however the free navigation system will be implemented, as the first aim of this project is to show the state-of-art in fairs.

7.1 Environments

Both systems are conceived to work in slightly different environments.

7.1.1 Magnetic tape navigation

The AGVs movements are heavily bounded in magnetic tape navigation, thus it is possible to move in narrow spaces as the trajectories are predefined. Furthermore the AGV is capable of localizing itself through RFID Tags on the ground and consequently change the security scanner's warning and alert zones in function of the environment.

Magnetic tape circuit are not needed to be circular, they can include dead ends. If a dead end is encountered a RFID Tag will change the movement direction of the AGV.

Circuits may include several routes in which branch divergences and convergences of appear. These routes are defined by destinations numbers. Usually AGVs are given a destination to reach. Before arriving to a branch deviation a RFID tag informs about the

destinations. The information contained is "If destination is X turn right/left, turn left/right otherwise". The magnetic sensor is then configured to follow the specified branch.

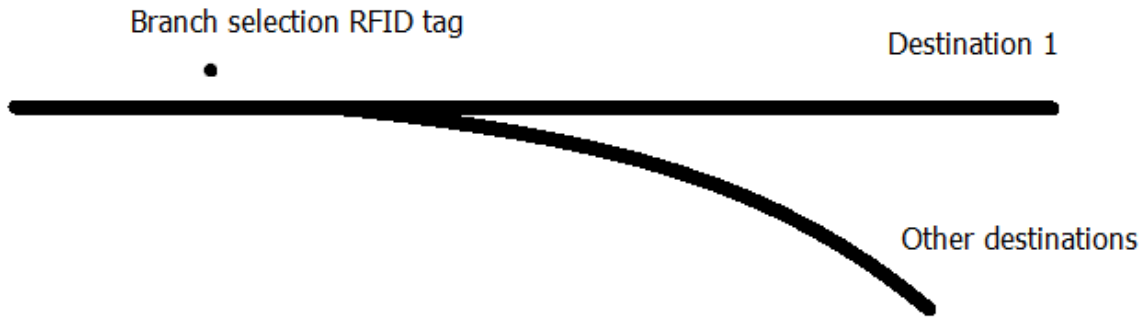


Figure 45: Example of branch selection. If the AGV is set to move to destination 1 it will follow the left branch, otherwise it will follow the right branch.

This system was conceived for fully automatic operation. However in the case of this project the user should be able to decide which branch is selected in every moment. One possible solution is analyzing the users position respect to the AGV in the x axis. If the position is negative it turns to the left, if positive to the right. However this could sometimes lead to erratic outcomes as shown in Figure 46: Example of erratic branch decision. The user is obviously on the left branch, however his x position respect to the AGV is positive, leading it to select the right branch..

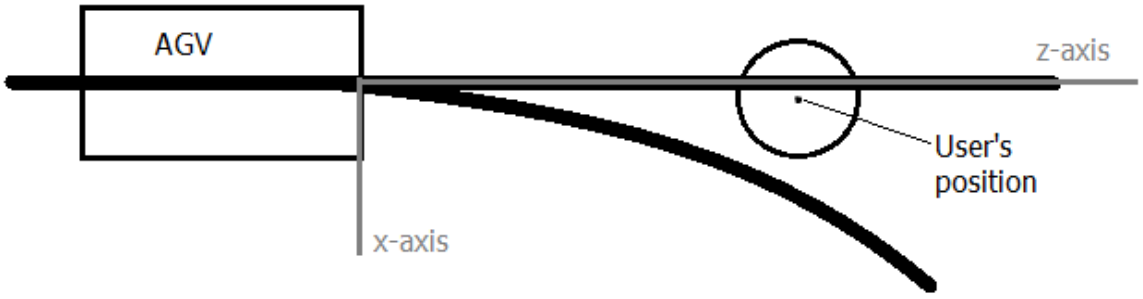


Figure 46: Example of erratic branch decision. The user is obviously on the left branch, however his x position respect to the AGV is positive, leading it to select the right branch.

7.1.2 Free navigation

Free navigation's environment could be any type of environment. The user must have in mind while guiding the AGV to not make it pass through narrow spaces as the security scanners may stop the AGV if they detect the walls.

7.2 Magnetic Sensor

Magnetic sensors for navigation are usually an array of actual magnetic sensors. A microcontroller receives the detection values of all sensors, computes the position of the centre of the magnetic tape and writes a voltage level according to the position on the output wire.

Figure 47: The magnetic sensor GS-2744B of the Japanese brand Macome Corporation. It contains an array of 15 small magnetic sensors [16].

In the magnetic tape navigation system the sensor will be placed in front of the AGV's wheels. In contrast in the free navigation system no sensor will be installed as the PC will emulate the sensor's output and inject them on the AGV's controller instead. An imaginary straight line is drawn from the user's position to the rotation centre of the front wheels and the point, in which this line intersects with the area where the magnetic sensor is, is considered as the point where the magnetic tape would be. The formulae to use are:

$$M_{mid} = \frac{(M_{max} - M_{min})}{2} \quad (13)$$

$$a = atan2(x, z) \quad (14)$$

$$x_m = \cos(a)\, d \quad (15)$$

$$m = \frac{(M_{max} - M_{mid})}{\frac{l}{2}} \quad (16)$$

$$output = M_{mid} + x_m * m \quad (17)$$

Where M_{max} and M_{min} are the maximum and minimum values the sensor provides, M_{mid} the middle value of the sensor, x_m the theoretical point where the magnetic tape would be, l

the length of the sensor, d the distance from the rotation centre to the sensor and x and z the coordinates of the user respect to the Kinect.

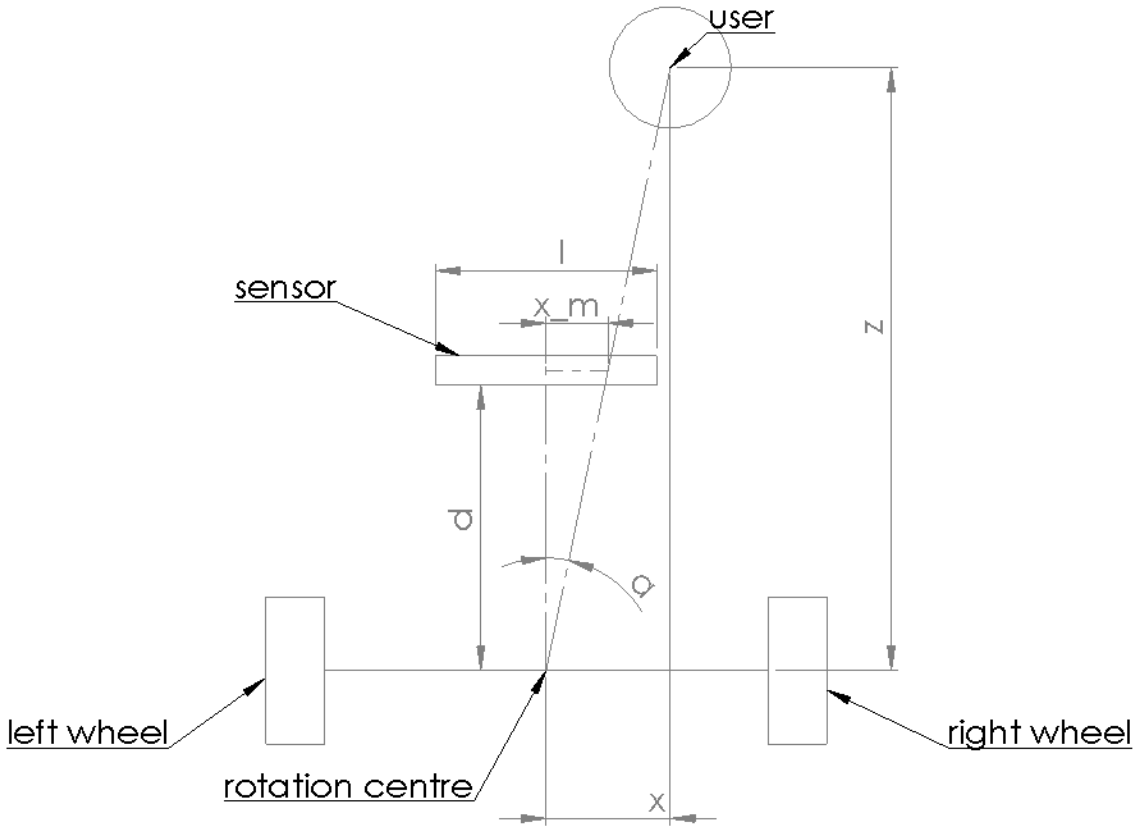


Figure 48: User position compared to the sensor's value respect to the rotation centre of the AGV's wheels. The Kinect sensor should be mounted on the rotation centre.

7.3 Velocity controller

The AGV's motors controllers are able to receive velocity set points and regulate it autonomously. The AGV should follow the user at a given distance with the same velocity as the user; therefore the velocity set point depends on distance to the user and the user's velocity.

The distance to the user can be obtained from the Kinect. Subtracting this distance from the desired distance gives the distance error. The system should react fast but little to distance changes; thus the error will be introduced in a PID or PD controller with a small P.

The velocity of the user respect to the AGV can be obtained deriving the distance. This velocity has to be added to the velocity obtained by the AGVs encoders to compute the user's global velocity.

The output of the PD controller is then subtracted from the user's velocity. The obtained velocity could be introduced directly into the motors drivers. However negative velocities are undesirable, thus zero saturation is introduced.

Either the PC or the PLC could compute this control algorithm. However the PC is chosen to perform them as it is more flexible while programming control loops.

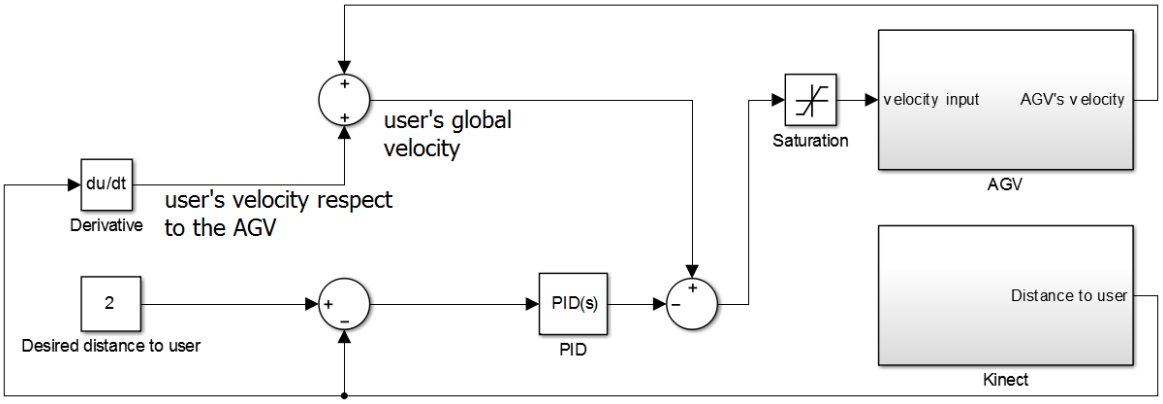


Figure 49: Controller scheme for AGV to control velocity and distance.

CHAPTER 8: GESTURES

Once the system has started it will commence to interact with possible users. If the system does not detect any human in front of it, it will stay on standby. However if it detects any human it will start searching for a user that wants to interact with it. To interact with the robot the user must perform the “engage” gesture. Once engaged the system will lock its attention to the user and follow him, even though another person does the “engage” gesture. To terminate interaction the “engage” gesture must be performed again. If the AGV was moving it will receive a stop order.

While engaged the system updates periodically the stored colour of the user’s t-shirt. If the user exits the field of vision a stop order will sent to the AGV and the system will start to search during an arbitrary amount of time a person with the same t-shirt colour. If it has not found the user in this time it will start searching for a user that wants to engage. If, while searching for the lost user, someone does the “engage” gesture the system will stop searching for that user and will engage to the new user.

Users can only give orders if the system has locked its attention to it. Orders are given through gestures. Gestures can only be done with the right arm. In the near future gestures with the left arm and with both arms will be introduced. While other projects have implemented a set of 3 gestures [52] here at least 7 different gestures will be used.

The following sections describe each gesture that has been added to the system database. Usability factors studies about effectiveness, efficiency and satisfaction [40] will be done in the "Results and Conclusion" section. If a gesture has low results it may be considered to be modified, exchanged or removed.

8.1 Engage

The engage gesture is not an order. It is the gesture done to start or stop the interaction. It will however give a stop order if the AGV is moving and the gesture is done to terminate interaction. The user must stretch his right arm up. It does not need to be fully stretched.

Figure 50: Engage gesture. Hold right arm completely up during 1 second.

8.2 Reset

The reset order must be given to rearm the AGV. The AGV unarms when it turns on or if it encounters an error like bumping into an object. The “reset” gesture is a very unnatural pose so the user must perform it consciously as it is a very uncommon and sometime dangerous order to give.

Figure 51: “Reset” gesture to rearm AGV. Hold the upper arm horizontally and the forearm vertically pointing up.

8.3 Come

The come order will activate the movement of the AGV. It is the first dynamic gesture introduced. It is the gesture with the biggest number of models, as each person does this gesture differently.

Figure 52: “Come” gesture. It is the common “come here” gesture.

8.4 Stop

The “stop” gesture will stop the AGV’s movement. This order will also be given when the user terminates interaction or the user exits the field of vision of the Kinect sensor automatically without the need of performing the gesture.

Figure 53: “Stop” gesture. It is the common “stay there” or “do not come closer” gesture.

8.5 Turn around

The “turn around” gesture is used to change the AGV’s movement direction. An imaginary parallel to de ground circle should be drawn with the hand, either clockwise or counter-clockwise. The AGV will start moving backwards. If the AGV has a rear Kinect, this one will be activated.

Figure 54: “Turn around” gesture. Perform a horizontal circle with the hand in either direction.

8.6 Silence

The “silence” gesture deactivates or activates the AGV’s horn sound. The horn sound is necessary when the AGV moves autonomously to alert other people that it is near. It can be extremely annoying when it is next to the user, so the possibility to deactivate it is given as it does not work autonomously while interacting with it. Once interaction is terminated activation order is given.

Figure 55: “Silence” gesture. The index finger is placed vertically in front of the mouth.

8.7 Turn right

The “turn right” does not give an order, though it will be recognized if performed. The gesture is implemented for testing purpose. In the future when left hand gestures are implemented it will be activated. It will probably be used to tell the AGV to navigate to the right of the user. In the system in which the AGV navigates on top of the magnetic tape it can be used to announce which deviation should be taken if it encounters one.

Figure 56: “Turn right” gesture. With the upper arm pointing down move the forearm up and down.

CHAPTER 9: VIRTUAL ASSISTANT

All interaction discussed till now was in the human-robot direction. However to accomplish a natural experience robot-human direction interaction must also be provided. For example if the user does a gesture the robot does not recognize the user must be informed of this, as instead he could think that the system is still trying to identify his gesture, ending in frustration.

As in the rest of this project this feedback should be the most natural possible. This includes:

- Simple and concise: the user should get only relevant information and avoid being bombarded with useless information.
- Immediate: the user should understand the shown information in the minimum time possible.

Therefore an image-based feedback seems an efficient way to achieve these requirements instead of a text-based one.

The most natural interaction for a human is interaction with another human, thus the author of this project has decided to use a virtual assistant that emulates human-human-interaction though facial expressions to maximize the user's comfort. It is said that 55% of the information of a communicative message is transferred by facial expressions. [38]

Several studies have come to the conclusion that feminine avatars are slightly more popular than masculine avatars. Females prefer in high rate female avatars and males are comfortable with both sex avatars. As well, more realistic avatars a preferred over cartoon avatars [30][31][32]. Nevertheless humans do not sympathize with too photorealistic avatars. This is known as the uncanny valley, term proposed by Masahiro Mori in 1970 [33][34].

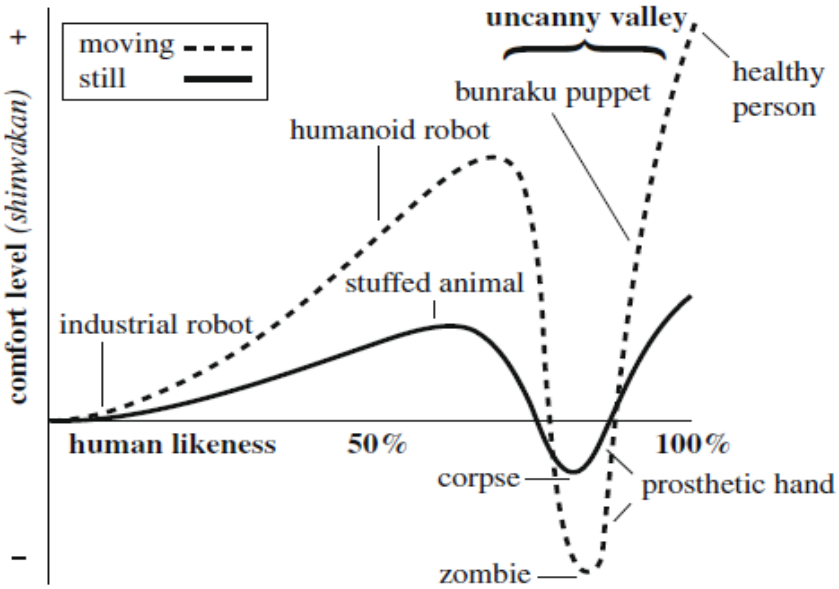


Figure 57: Graph proposed by Masashiro Mori in 1970 in which the "uncanny valley" or "bukimi no tani" can be appreciated.

As the enterprise in which this project is developed does not have a graphical design department, the decision was taken to use an androgynous simple cartoon like avatar inspired by the Baxter robot, an industrial robot with a screen showing a face that changes the shown emotions in function of its states.

Figure 58: The Baxter robot communicates with the user through the virtual assistant in his screen.

For this project also each state will be represented by an emotion.

9.1 Standby

While the AGV does not detect any human it will stay in standby. The face will appear as sleeping.

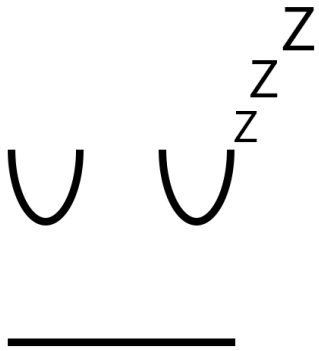


Figure 59: Standby face. The AGV is “sleeping”.

9.2 Error-free

The “error-free” face shows up when the AGV has no error, is tracking a user (or searching for it) and stopped. The AGV is waiting for new orders. It is a neutral, slightly smiling face.

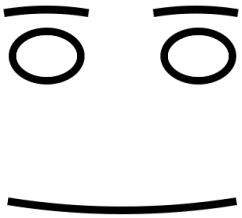


Figure 60: “Error-free” face. Neutral emotion, slightly smiling.

9.3 Moving

The “moving” face appears when the AGV is moving. It looks happy and has sunk a little bit its eyebrows as it would be concentrated.

Figure 61: “Moving” face. It looks happy and has sunk a little bit its eyebrows as it would be concentrated.

9.4 User lost

The “user lost” face appears when the user exits the field of vision and the system searches for him. An interrogation sign appears on its head as it would be confused. It looks slightly sad and has bowed its eyebrows.

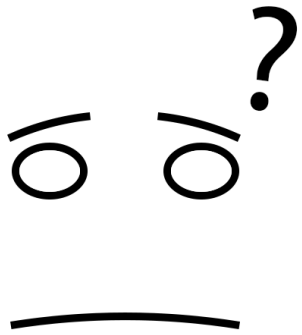


Figure 62: “User lost” face. It looks confused and slightly sad.

9.5 Error or warning

The “error or warning” face appears when an error or warning has occurred. It looks slightly sad and shocked as its eyes are bigger and its eyebrows higher than usual. Gong showed that users trust more a positive looking avatar than a negative looking avatar [35], therefore the emotion is only slightly negative.

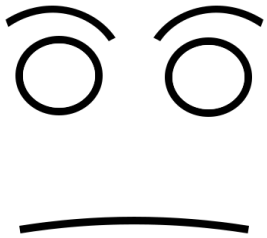


Figure 63: “Error or warning face”. It looks slightly sad and shocked.

9.6 Eyes

The faces shown till now have no irises. However they indeed have moving irises. In contraposition to the Baxter robot, who "looks" at the piece he is manipulating, this system looks to the user, improving the interaction feeling. If a user is being searched, the eyes will move around giving the feeling that the system is actually searching for someone.

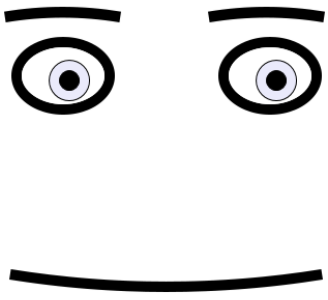


Figure 64: The eyes will move freely while searching for a user or will focus on the user if he is tracked.

9.7 Gesture recognized

The face emotions change depending on the status of the AGV. If the user gives the order to move the gesture first needs to be identified, then the order is given through the communication channel to the AGVs controller, the controller needs to interpret it, change its status and finally inform through the communication channel of its status. Only then the face emotion is changed. This process is usually almost immediate but it could last more time. To inform the user that the system has at least identified the gesture it will show a red exclamation sign on top of the face during a short period of time.

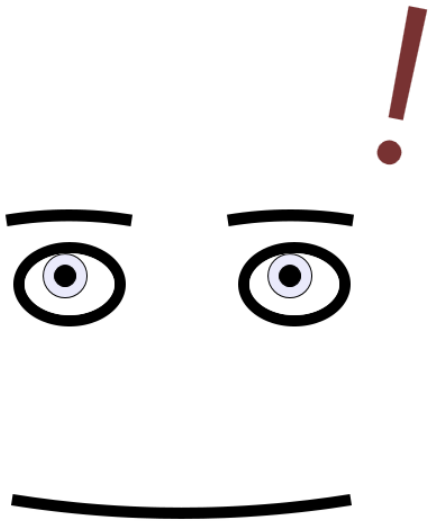


Figure 65: Once a gesture is identified an exclamation sign appears on top of the face during a short period of time.

CHAPTER 10: GRAFCET REPRESENTATION

GRAFCET (Graph Fonctionnel de Commande Etape-Transition) is a theoretical synthesis of different tools for system design. It was developed in 1977 by the French association AFCET (Asociación Francesa para la Cibernética Económica y Técnica) as a graph to describe system automations. [44] It consists of only 3 different elements [45]:

- Phases: represented as a box. Each phase has some actions associated to it.
- Transition: represented as a horizontal short line. Each transition has associated an condition in which the transition occurs.
- Directed unions: unions between phases and transitions and transitions and phases.

Phases can be active or inactive. A phase can only be activated if the previous phase is active and the conditions of the transition in-between are met. Once this occurs the previous phase switches to inactive.

Sometimes the next phase can be any of a list of phases, depending on the transition conditions. These possible phases are represented in parallel connected with a horizontal line.

In other occasions 2 o more phases have to be activated at once. These phases are also represented in parallel but connected with a double horizontal line.

There exist 3 levels of GRAFCET [44]:

- Descriptive: It uses natural language to describe each phase and transitions. It is easy to understand.
- Technologic: Phases and transitions are described in a more technical way, showing which kind of technology will be used.
- Detailed: SFC (Sequential function chart). Implementation level description. This level could be used as a direct programming language for PLC.

This project involves different technologies (PC and PLC) communicating between them. How they work internally has no relevance, besides the PLC program was already designed and has suffered only slight alterations for communications. Therefore the GRAFCET has been designed on descriptive level. Its purpose is to give an general idea

on how the whole system works. The GRAFCET has not security issues into account. These will be discussed and added in the following chapter.

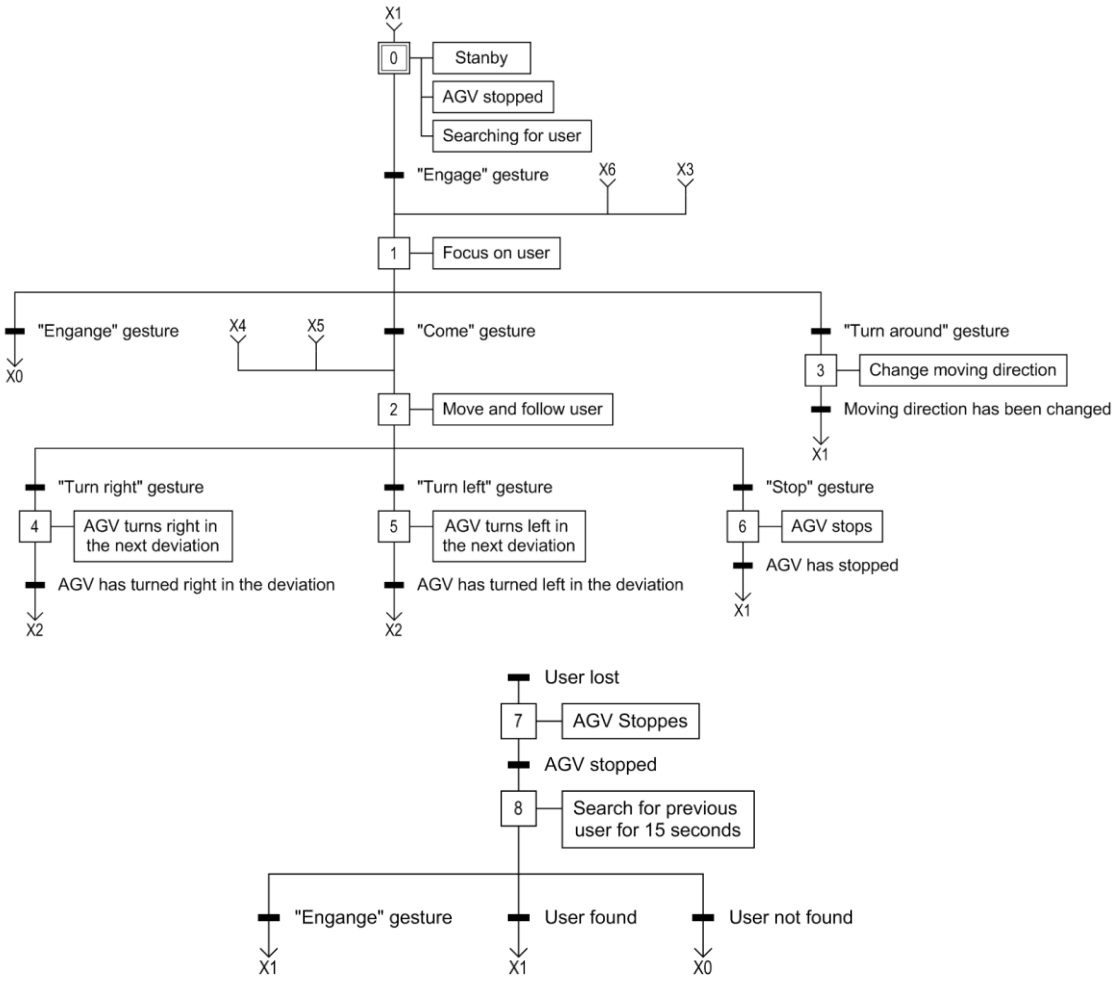


Figure 66: General GRAFCET of the system

CHAPTER 11: SECURITY

Usually the AGV should work without failures. However it may be possible that the system fails. Those failure could take place due to various reasons:

- The AGV has bumped into an object or has detected a object too near to itself
- The battery is low
- The motor drivers have failed
    - Because the voltage is to low
    - Because the energy consumptions has been to high (motor is stuck)
- The AGV has lost its track (only in magnetic tape navigation)
- The PLC has failed
- Communication system has failed
- Internal electronics have failed

If a failure occurs the AGV stops whichever the origin is. The start and stop process has been developed using the GEMMA guide (Guide d'Etude des Modes de Marches et d'Arrêts) (Design guide to start up and stop modes).

11.1 Introduction to the GEMMA guide

Production systems present nowadays an high grade of complexity in their automation design. Therefore methods to ease the development to accomplish a guaranteed safe system are needed. This methods should not only deal with the normal function of the system but also stopping and start up processes, semiautomatic and manual modes, and emergency and failure treatments. Thus the French agency ADEPA (L'Agence Nationale pour le Développement de la Productique Appliquée à l'Industrie) developed in 1984 the GEMMA guide. [42]

The aim of the GEMMA guide is to provide a GRAFCET similar graph that includes all basic possible states to be used as a base model. The designer should cross out the states that will not be necessary in his system, and erase or add transition arrows as necessary from state to state. Finally the conditions for each transition to be fired need to be added. If a transition is performed automatically after the actions of the states are completed it is not necessary to add a condition.

With the obtained graph the designer can then design the GRAFCET, making sure he will not miss any process.

The GEMMA guide divides the whole process in 4 main processes, which are "control without power", "stop procedures", "operating process" and "fault process". In addition

there is a parallel process that encloses part of the three latter called "production". The main processes are divided in other sub-states. [43]

Stop procedures sub-states:

- A1 Stopped at initial conditions: The machine is idle in its initial state condition.
- A2 Stop after cycle: Transient state in which the machine (which was working normally till now) must stop finishing the cycle and end up in A1 state.
- A3 Demanded stop: Transient state in which the machine must stop in a given condition to end up in A4 state.
- A4 Obtained stop: The machine is idle but not in its initial state condition.
- A5 Preparations to start after failure: State in which all required manoeuvres to start after a failure are done (e.g. emptying, cleanup,...)
- A6 Initial state preparations: State in which preparations to achieve the initial conditions are performed.
- A7 Specific state preparations: State in which preparations to achieve a given conditions are performed.

Operating process sub-sates:

- F1 Normal production: State in which the machinery produces normally. This state performs the duties for which the machinery was designed for.
- F2 State of preparedness: State in which the required manoeuvres to start the production are performed (e.g. warm-up)
- F3 State of completion: State in which the required manoeuvres to restore the initial conditions after completion of the tasks are performed
- F4 State of verification without order: State in which the machinery can be controlled partially or completely by an operator. It is the so called "manual control" mode and it is used for maintenance or verification.
- F5 State of verification with order: In this state the machinery works normally but in the pace the operator dictates. It is the so called "semiautomatic control".
- F6 Test status: State in which adjustment or maintenance operation are performed.

Fault process sub-states:

- D1 Emergency stop: State achieved after an emergency stop.
- D2 Failures treatment: State in which the failures can be diagnosed with or without the operator.

- D3 Production even with failures: State in which the machinery still works even though failures have appeared.

Figure 67: Graphical representation of the GEMMA guide represents graphically with arrows the transitions a state can perform to achieve another state. The arrow that points to the emergency stop states has no origin; this means that it can be accessed by any state.

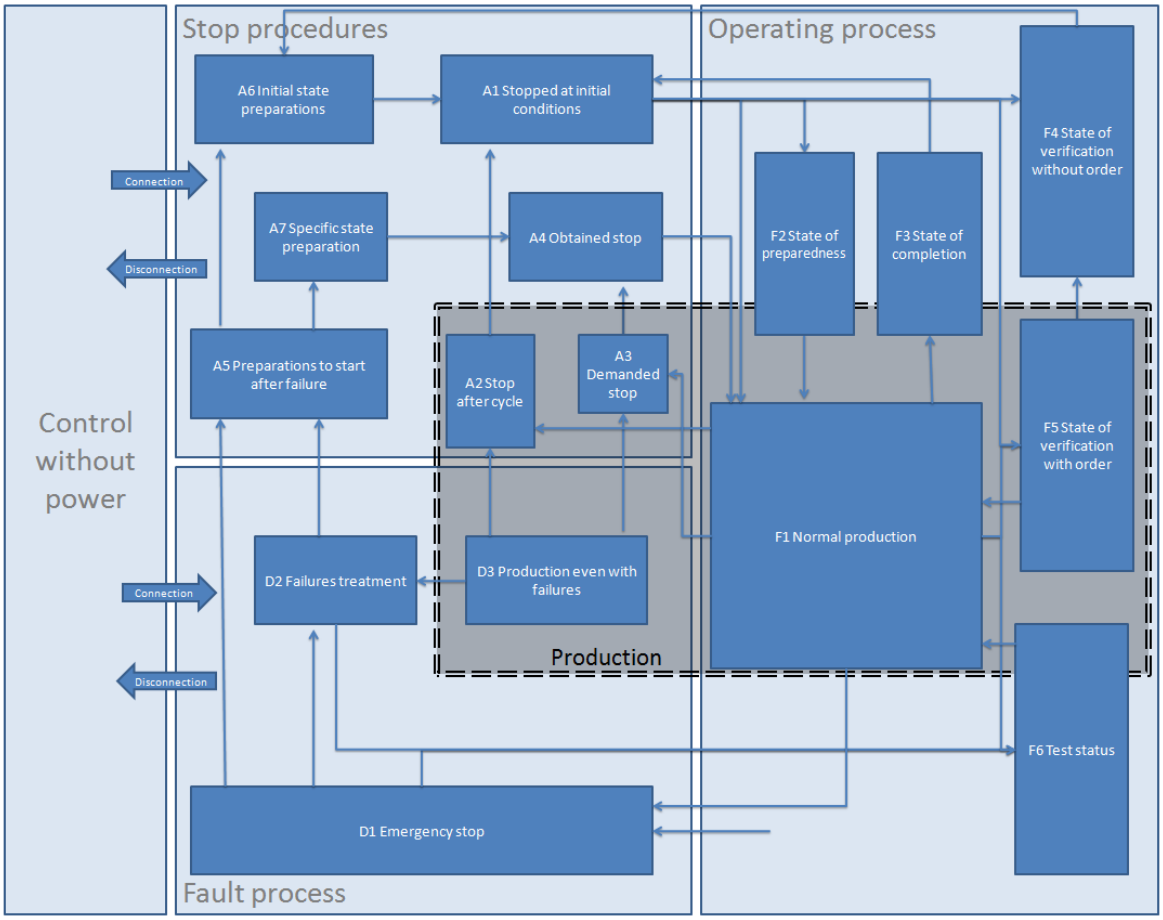


Figure 67: Graphical representation of the GEMMA guide

11.2 GEMMA guide applied to the AGV

This section discusses how the GEMMA guide is applied to the AGV's PLC and explains every process. Some procedures have been erased, as also some transition. However other transitions have been added.

- A1: The AGV is stopped and brakes are activated.

    - A1 to F2: The user has pressed the "go/stop" pushbutton or made the "go" gesture.
    - A1 to F4: The user has connected the joystick controller and pressed the "go/stop" pushbutton.
- A2: The AGV deactivates the march and activates de brakes.
    - A2 to A1: This transition is performed automatically after the breaks have been activated.
- A3: Does not exist.
- A4: Does not exist.
- A5: The AGV still displays a general error state, even though the failure was corrected.
    - A5 to A6: The technician has pressed the "reset" button or done the "reset" gesture.
- A6: The internal security electronics are reset.
    - A6 to A1: This transition is performed automatically.
- F1: The AGV is working normally, following the user and performing his orders.
    - F1 to A2: To user has pressed the "go/stop" pushbutton or performed the "stop" gesture.
    - F1 to F5: A technician has activated the diagnosis modus.
    - F1 to D3: The battery is low.
- F2: The breaks are deactivated and the AGV starts to move.
    - F2 to F1: This transition is performed automatically.
- F3: Does not exist.
- F4: The technician can control the movement of the AGV with the joystick controller.
    - F4 to A6: The technician has disconnected the joystick controller.
- F5: The diagnosis modus is active. The technician is able to see some parameters on the screen instead of the face.
    - F5 to F1: The technician has deactivated the diagnosis modus.
- F6: Does not exist.
- D1: A failure has occurred (excluding low-battery error).
    - D1 to A1: If the failure was a communication error the transition will be performed automatically once the communications is re-established.
    - D1 to D2: If the failure was because of any other reason the transition will be automatic.
- D2: The AGV waits till the failure has been fixed.
    - D2 to A6. If the error was due to the security scanner the transition will be automatic once the error is fixed.

- D2 to A5: Else, once the error has been fixed this transition will be automatic.

- D3: The AGV moves to a charging stations.
  - D3 to A2: Once the AGV arrives to the charging stations the transition will be automatic.

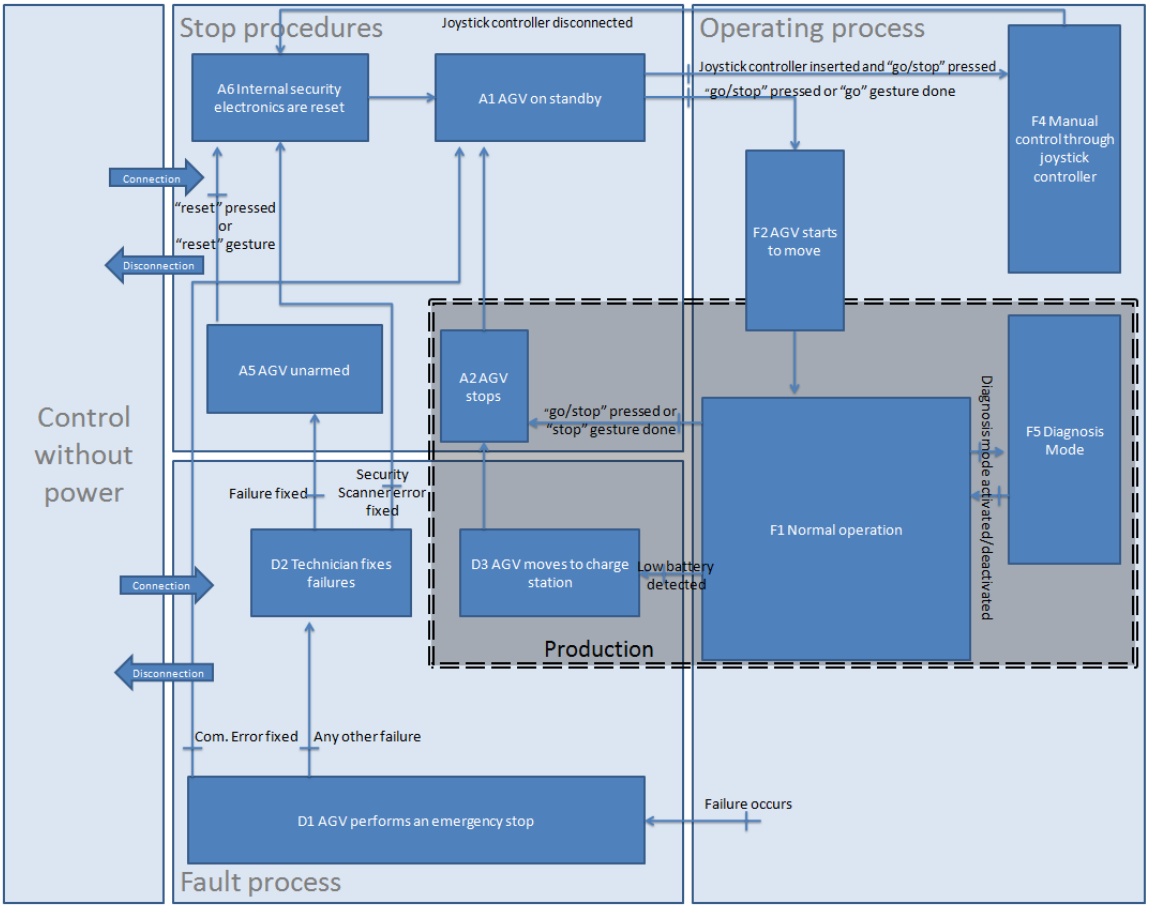


Figure 68: GEMMA guide representation applied to the AGV system

As a result of using the GEMMA guide two new branches appear in the GRAFCET diagram. Both of them are parallel to the till now designed GRAFCET. The branch AB can only be activated while the AGV is moving. The CDE branch can be activated anytime. As a result of this conditions branch CDE has a higher priority over AB. Normal activity can only be restored once both branches have been completed.

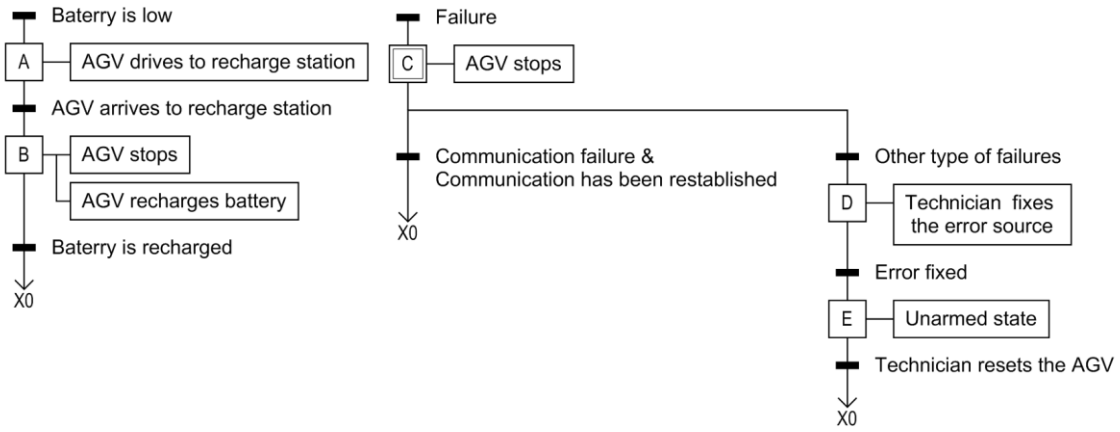


Figure 69: Failure and security branches of the GRAFCET

11.3 AGV operator

Two types of AGV operators have been mentioned: the user and the technician. The user has a basic knowledge of how the AGV works, enabling him to use it for normal production. If an error occurs the user should not fix the problem. Instead a qualified technician, who has in-deep knowledge of the AGV, should fix it.

Operator Type	Knowledge about AGV	Interaction with AGV
User	Low	Engage, start, stop, turn left/right, turn around
Technician	High	Same as user + Reset + Use of joystick controller + Diagnosis mode

Table 1: Possible AGV users

CHAPTER 12: ASSEMBLY

The final step of the design involves the physical aspects of the system. First of all different GUIs (Graphical User Interfaces) have to be designed. The PC and Kinect sensor need to be powered on, therefore there is a need for a specific power source PCB (printed circuit board). A power controller is also needed to send a power on signals and power off signals to the PC at the same time as the AGV starts or stops.

12.1 Main GUI

The philosophy in which the main GUI was designed is "simplicity". The user does not need any more information than the one that the avatar provides and should also not be bombarded with technical information. This is the reason why the GUI is a blank screen with the avatar in middle. In the lower left part of the window a little text informs if the application is "Running" correctly or if an "Error" occurred. Hopefully this will decrease the distraction sources for the user.

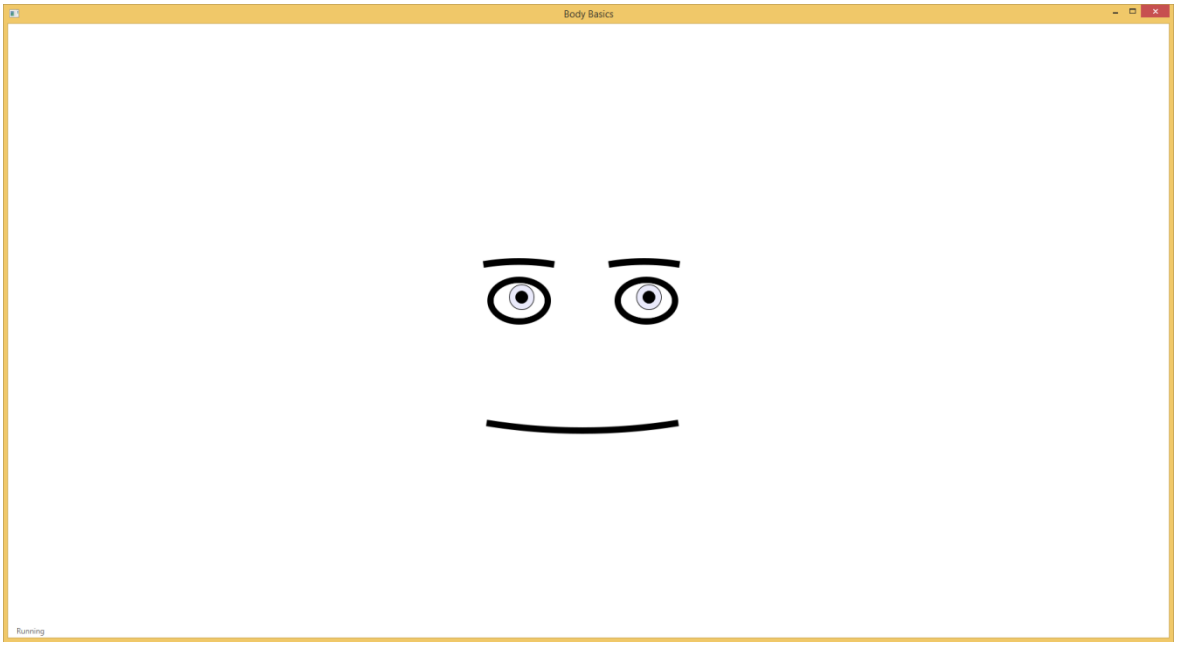


Figure 70: Normal graphical user interface.

Sometimes there could be the need for more information, for example while implementing new gestures or identifying error/failure sources. The GUI can be modified to show the needed information. This is called the debugging mode. The debugging mode can show information about de gesture recognition system or the AGV. Information about the gesture recognition system is:

- Recognition levels: shows a progress bar for each stored gesture. The progress bars fill in function of how likely the performed gesture is the respective gesture. If

the likeliness is high enough to be considered as the pretended gesture, the colour of the progress bar changes.

- T-Shirt colour: it may occur that due to lighting conditions the T-Shirt colour recognition does not work properly. A little coloured square shows the colour the system searches for.

Information about the AGV is:

- Battery level: Shows a battery icon with the percentage of left battery.
- Error code: Text showing all present error codes.
- Deviation: Informs about the next deviation direction the AGV will take.

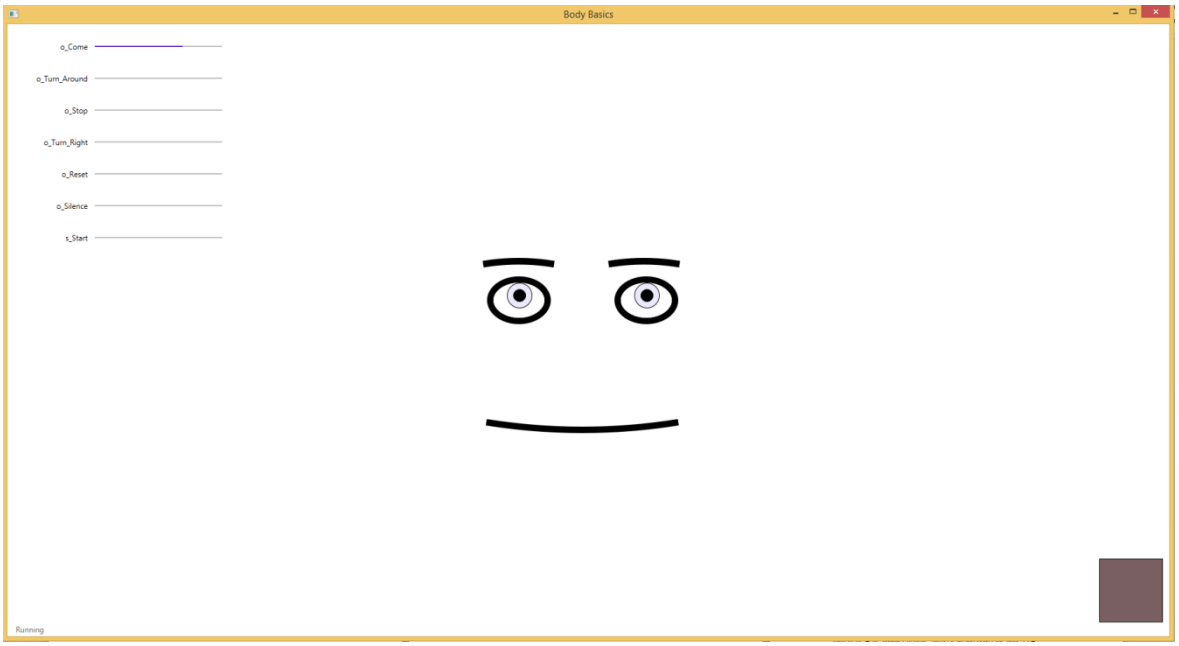


Figure 71: Debugging graphical user interface for gesture recognition system.

In both GUI types the user can not interact using the keyboard or mouse. The system is supposed to be mounted on the AGV, so there will not be any typical PC interface available. The user/technician can only interact through gestures or the onboard control panel of the AGV.

12.2 Gesture builder

The Gesture builder is separate software to define new gestures. It is not a WPF application, but a Windows RT application. The reason for this is that, as stated in chapter

3, the whole project was first developed to be used as a Windows RT application, but due to the unavailability of serial port interfacing it was migrated to a WPF application. The Gesture builder was developed beforehand this migration, and, as it does not need serial port communication, it has not been migrated.

The Gesture Builder is not thought to be used on the AGV. Instead it should be used on a Windows 8 PC in an office, lab or wherever the technician wants and is able to connect a mouse and keyboard interface.

The GUI is considerable more complex than the main GUI. It presents a black large square in the middle. This represents the field of vision of the sensor. If a user is detected its skeleton will be displayed. 6 users can be detected at the same time. As there is no "engage" gesture to focus on a user, a checklist on the right side allow to select the user. If only 1 user is detected the focus will be automatically on it.

A dropdown list is present on the right top side of the screen. The technician is able to select a gesture he wants to modify. The software loads the list of a database. If no database exists the software creates one, however the list will be empty. To add new gestures a menubar is placed in the bottom. Several buttons are available:

- Add: Adds a new gesture. A little window shows up asking for a name for the new gesture.
- Remove: Removes the selected gesture of the gesture list
- Delete: Deletes the last recorded gesture of the selected gesture. For each gesture several gestures can (or should) be recorded.
- Save: Overwrites the database.
- Edit: A little windows shows up enabling the change of name of the selected gesture.

On the left top side is a checkbox. Once the technician is ready to record gestures he should check that checkbox. To record a gesture the key "R" must be pressed during the whole recording process. A gesture has a length of 1 second or 30 frames. If the "R" is released before the 30 frames, the gesture must be deleted as the last frames will contain erroneous information.

Under the checkbox a table appears with each angle of each frame. Frames are represented in rows. The columns correspond to: Shoulder yaw angle, shoulder pitch angle, shoulder roll angle, elbow pitch angle, elbow roll angle, hand pitch angle.

Under the table a text informs about how many models have been recorded for this gesture.

Finally in the bottom information appears about the smoothness of the last recorded gesture. If it says it is not smooth the model should be deleted.

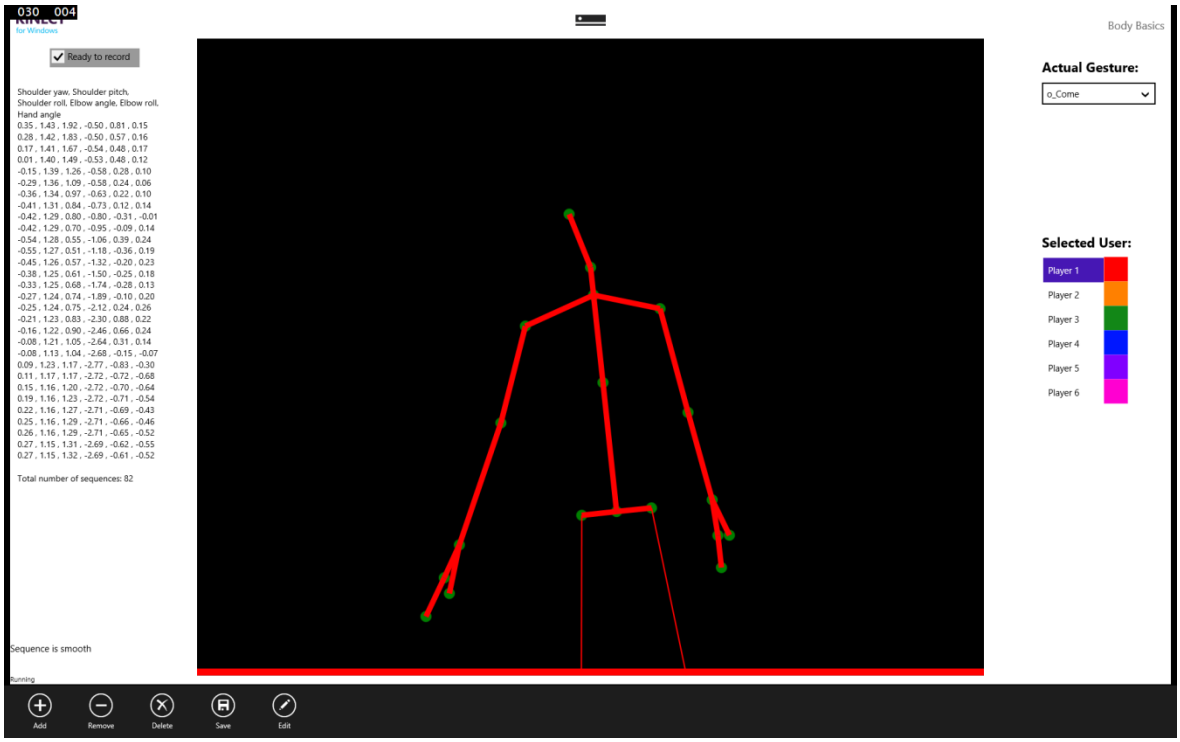


Figure 72: Gesture builder graphical user interface.

12.3 Power supply

The PC can work with a voltage source of 12 V to 18V and it may consume up to 3A. There is no information about the power source the Kinect sensor needs, but the power adapter that is sold with the sensor is a 220V to 12V/2,7A source. As it transform 220V alternating current it cannot be used on the AGV, as the AGV works with a 24V direct current battery. Therefore the solution is to build a power supply that converts 24V into an 12V/6A source. The use of linear regulator are not advised as they present a varying efficiency depending on the input/output voltage rate.[46] As this rate is 24V/12V the efficiency would be very low (around 50%). Instead switching-mode power supplies will be used. These present a very high efficiency (over 90% [47]). The drawback is that they need more complex electronics than the linear regulators.

The LM2676-ADJ regulators have been selected to be used. The output voltage of this regulator can be adjusted between 1,2V to 37V through a pair of resistances. They offer a maximum of 3A output, thus 2 of them will be used in parallel. The outputs will be connected by two diodes to avoid sink current problems. Diode usually have an voltage drop of 0,7V therefore the regulators will be designed to supply 12,7V.

The AGV has already a 10V and a 5V power source. These power sources are mounted in a Phoenix Contact's ME MAX 22,5 3-3 KMGY [48] housing offering a electric bus on the bottom. The new power supply will also be mounted in this housing to receive the battery voltage through the bus and avoiding further wiring.

Figure 73: Phoenix Contact's ME MAX 22,5 3-3 KMGY housing

Schematics and PCB layout can be found in the annexes.

12.4 Power controller

The Intel NUC PC offers a pin header to simulate electronically the power button. This is very convenient as the user should not worry about powering on or off while powering on or off the AGV. Instead thanks to electronics this can be done automatically. The behaviour should be:

- If the AGV is turned on, the PC is off and the battery voltage level is over a threshold, then the PC should be turned on.
- If the AGV is on, the PC is off and the battery voltage level rises over the threshold, then the PC should be turned on.
- If the AGV is turned off and the PC is on, then the PC should be turned off.

- If the battery voltage drops under a threshold and the PC is on then the PC should be turned off.

The signal to power on and power off is exactly the same, 2 contacts of the NUCs pin header must be short-circuited for a given period of time. This is why the system is not only depended on the input variables, but also on a clock and on the previous state (once the signal has given, it must not be given again until the input changes back and forth). Thus synchronous sequential logic has been applied. Voltage comparisons haven been implemented with operational amplifiers and "and" and "or" functions with standard logic gates.

The most complex part is the one that makes this system sequential instead of combinational. To create the "turn on/off" signal a circuitry is needed that outputs a logic "1" during a time only when the input rises from "0" to "1". 2 D flip-flops, an astable timer and two "and" logic gate have been used to create this signals.

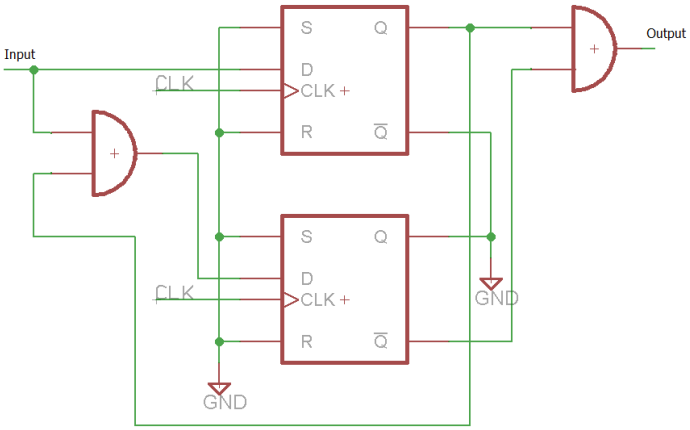


Figure 74: "Turn on/off" signal generating circuitry

Analyzing the circuit:

Input	D1	Q1	D2	/Q2	Clock	Output
0	0	0	0	1	0	0
0	0	0	0	1	1	0
1	1	0	0	1	0	0
1	1	1	1	1	1	1
1	1	1	1	1	0	1
1	1	1	1	0	1	0
1	1	1	1	0	0	0
1	1	1	1	0	1	0
0	0	1	0	0	0	0
0	0	0	0	1	1	0
0	0	0	0	1	0	0
0	0	0	0	1	1	0

Table 2: State transitions analysis of "Turn on/off" signal generating circuitry, in green states that have changed.

As seen in the table the output maintains in 0 if the input maintains also in 0. Once the input rises to 1 the output changes to 1 for one clock cycle and goes back to 0. The whole system goes back to its initials conditions one clock cycle after the inputs goes to 0.

The PCB is mounted again in a Phoenix Contact's ME MAX 22,5 3-3 KMGY housing. The schematic and PCB layout can be found in the annexes.

CHAPTER 13: RESULTS AND CONCLUSIONS

The designed system must be trustworthy in order to be implemented in a real environment. As the system has not been still installed on an AGV the test has been performed in the office with an Arduino microprocessor simulating the AGV's PLC. The gesture database has been constructed with 8 different persons each of them performing 30 samples of each of the 7 gestures. That is 240 samples per gesture and 1680 samples in total.

The testing has been performed with 5 persons. 2 of them have also participated in the database construction. Each of them has been asked to perform every gesture 20 times.

13.1 Confusion matrix

The results are shown in the next confusion matrix where the vertical axis is the performed gesture and the horizontal axis the recognized gesture.

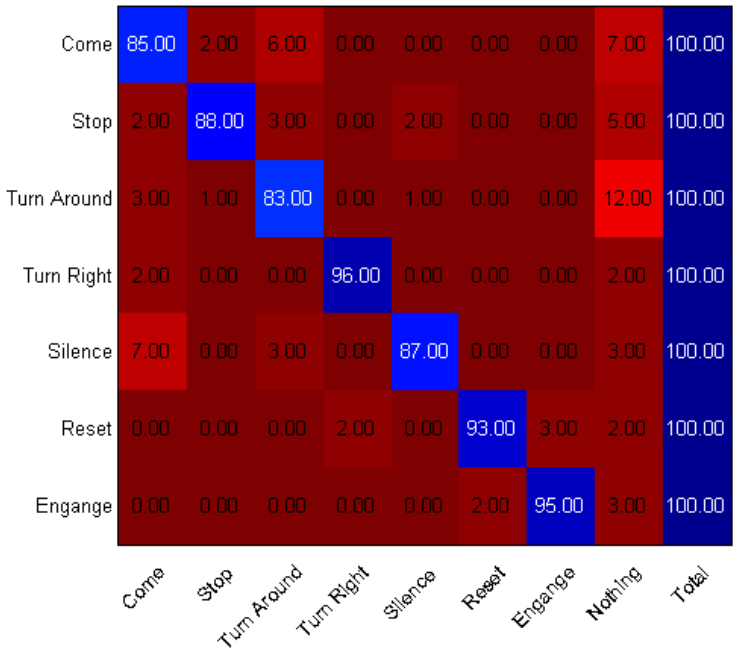


Figure 75: Confusion matrix of the whole dataset

It is of interest to see how the confusions matrix changes for people who have participated in the database construction and people who did not.

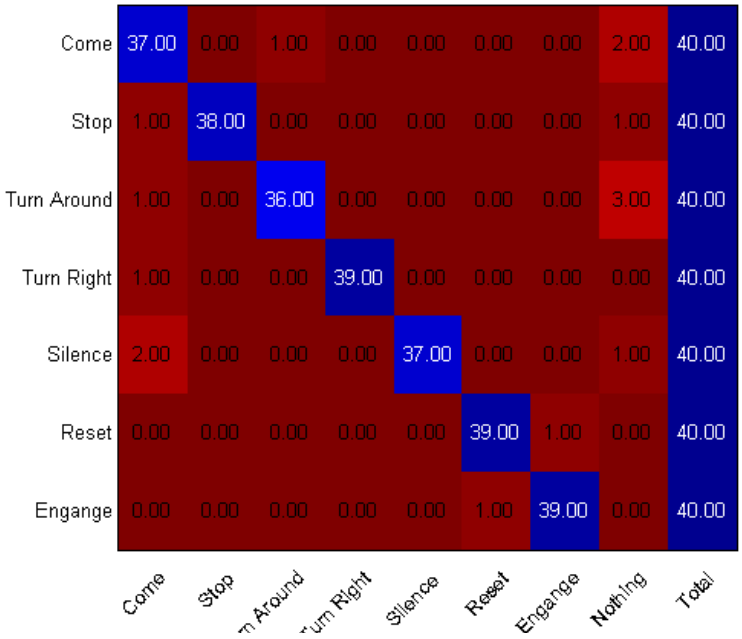


Figure 76: Confusion matrix of dataset performed by people who participated on the database construction

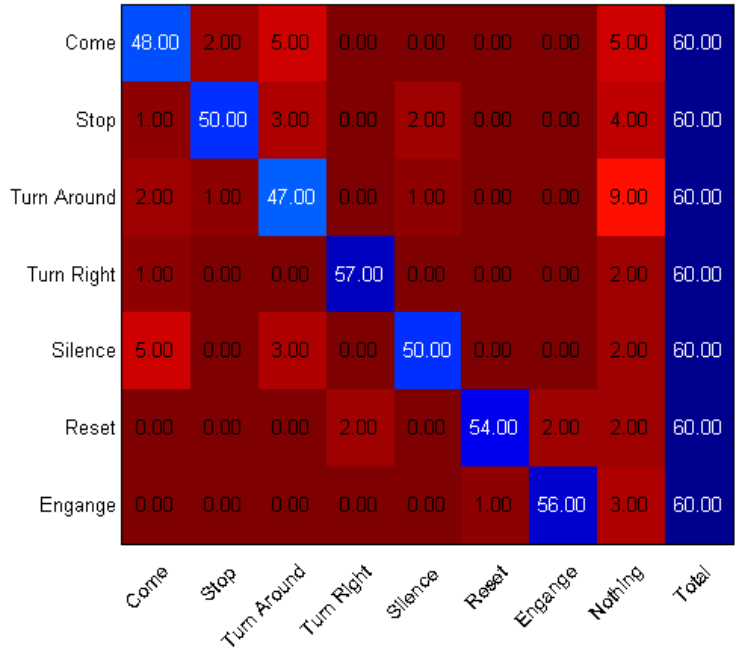


Figure 77: Confusion matrix of dataset performed by people who did not participate on the database construction

As seen the performance of the users who did participate in the database construction is notable higher. This is due to two main reasons:

- As they have performed the gestures several times prior the test they are "trained", they know how to perform the gesture correctly.
- Each individual carries the gesture out in a slightly different manner. The database has explicitly stored the uniqueness of these 2 users; therefore it is more adapted to them.

The most misinterpreted gesture is the "come" gesture. This could lead to not desired behaviours of the AGV. Nevertheless its performance is still high so it will not be changed.

The "reset" gesture has one of the highest performance which is desired as it is a critical order because of security reasons.

13.2 User's experience

Furthermore the users have been asked to answer a survey. It consist of 12 phrases in which the respondent should decide if he is strongly agrees, agrees, is neutral, disagrees or strongly disagrees using the Likert scale [55].

The phrases and it results are:

Strongly agrees 4	Agrees 3	Neutral 2	Disagrees 1	Strongly disagrees 0	Total score
1) Controlling the AGV with gesture is intuitive.					
1	2	2			2.8
2) Learning to control the AGV with gestures is fast.					
1	3	1			3
3) Gestures are easy to remember.					
4	1				3.8
4) Gestures are easy to perform.					
3	1	1			3.4

5) Gestures do not cause fatigue.					
1	3		1		2.8
6) The gesture recognition system does not frustrate the user.					
	3	1	1		2.4
7) It is not necessary to show the recognition level bars.					
	1	2	2		1.8
8) The avatar on the screen transmits understandably the state of the AGV.					
	3	1		1	2.2
9) The avatar conveys confidence.					
	2	2	1		2.2
10) The whole system conveys confidence.					
	2	3			2.4
11) I'm willing to work with this system.					
	3	1	1		2.4
12) I think this system could be useful for industry.					
2	2	1			3.2

Table 3: Questions the users answered and results.

From these results the following conclusions can be obtained (as the users did not interact with an actual AGV but a simulation, phrases 1 and 2 can be considerate to have no relevant information to be analysed):

- It seems gestures are easy to remember and perform and generally do not cause fatigue.
- Users can sometimes get frustrated working with this system although it is not usual.

- Users consider that the recognition level bars are highly useful to not getting frustrated.
- The avatar usually transmits understandably the state of the AGV, although some users do not understand the avatar's emotions.
- The trust level of the users into the avatar/system is not as high as expected.
- Most of the users think they would want to work with such a system.
- Users are optimistic about the usefulness of the system.

13.3 Effectiveness, Efficiency and Satisfaction

With the results presented a study about effectiveness, efficiency and satisfaction can be done. The guide by Cristina Manresa-Yee about usability of vision-based interfaces [40] has been used.

13.3.1 Effectiveness

The effectiveness studies how accurately and completely the user achieved the goals. It can be calculated as number of correctly recognized gestures divided by the total number of performed gestures. The effectiveness of each gesture is:

- Come: 0.85
- Stop: 0.88
- Turn around: 0.83
- Turn right: 0.96
- Silence: 0.87
- Reset: 0.93
- Engage: 0.95

As seen the effectiveness of each gesture is pretty high.

13.3.2 Efficiency

Efficiency is the relation between effectiveness and the amount of effort performed by the user. It can be divided in 4 attributes which are all (except for "duration" calculated subjectively through questionnaires.

- Physical fatigue is obtained from phrase 5 and has a score of 2,8/4.

- Duration is how long the user needs to perform the gesture. All gestures have a duration of 1 second which is pretty low.
- Cognitive load is the total amount of mental activity imposed on working memory. Its score is obtained from phrase 3 and 6. It is 3,1/4.
- Learnability and Memorabilty is the time it takes for the user to learn the gestures. In the case of this project, instead of measuring it by time it is measured by phrase 3. The score is 3,8/4.

As seen the efficiency is generally high.

13.3.3 Satisfaction

Satisfaction involves how comfortable the users feel while using the system. It may divided in 5 attributes, evaluated, again, through questionnaires.

- Naturalness and Intuitiveness is how real the interaction feels. Its score can be obtained of phrases 1 and 7. It is 2,3/4.
- Comfort is how well the user feels while using the system. A score of 2,8/4 is obtained from phrases 4, 5 and 6.
- Ease of use is self explanatory. A score of 3,6/4 is obtained from phrases 3 and 4.
- User experience and Satisfaction of use is how satisfied the user feels while interacting with the system. It can be obtained from phrases 9, 10 and 11. The score is 2,2/4.
- Social acceptance is how much the user has accepted the system. A score of 2,8/4 is obtained from phrases 11 and 12.

Satisfaction is not as high as efficiency but can be still considered high.

13.4 Achieved objectives

Finally these objectives were achieved:

- The system is able to focus on a specific user, ignoring other persons in the field of vision.
- The system is able to recognize and interpret satisfactorily the gestures performed by the user.
- The system is able to memorize the user’s t-shirt’s colour in order to recognize him again if he exists momentarily the field of vision.

- The system is able to follow a user at a safe distance. This is not been proven empirically but theoretically.
- The DTW algorithm has been improved to raise the gesture recognition liability.
- The system has been proved to be usually natural.
- The system has been designed to be ready for being implemented on an AGV.

CHAPTER 14: FURTHER WORK

Far from being a mature, this project can be improved further. Some of the possible improvements are:

14.1 CAN Network

Lately the company has adapted its AGVs to use a CAN network to communicate PLC with motor drivers instead of using analogue wiring. This network could also be used to communicate the mini-PC with the PLC instead of using the RS232 network. The CAN network provides several benefits against the RS232 one [56]:

- Message priority system
- Can be a real-time system
- Does not need message collision avoidance
- CAN has an advanced error management
- CAN protocol is implemented in hardware, freeing load on the CPU
- Message filtering available

14.2 Left arm

The system developed in this project is only able to detect gestures performed with the right arm due to simplicity reasons. This is a disadvantage for left-handed users. Therefore the system will be improved by analysing also the left arm. This does not only let users perform gesture with the left arm, but also perform gestures in which both arms are involved. The greatest difficulty would be to interpret if a user has performed a gesture with one arm or both.

14.3 Best n sequences mean system

Right now the gesture recognition is performed in basis of the sequence with highest DTW score of each gesture set. This could lead to some trouble. For instance a user is performing the “come” gesture which was very similar to 50% of the model sequences of the same gesture, but it was also very similar to only 1 sequence of the “stop” gesture. If this single “stop” sequence has a higher score than any of the “come” gestures it will be identified as a “stop”.

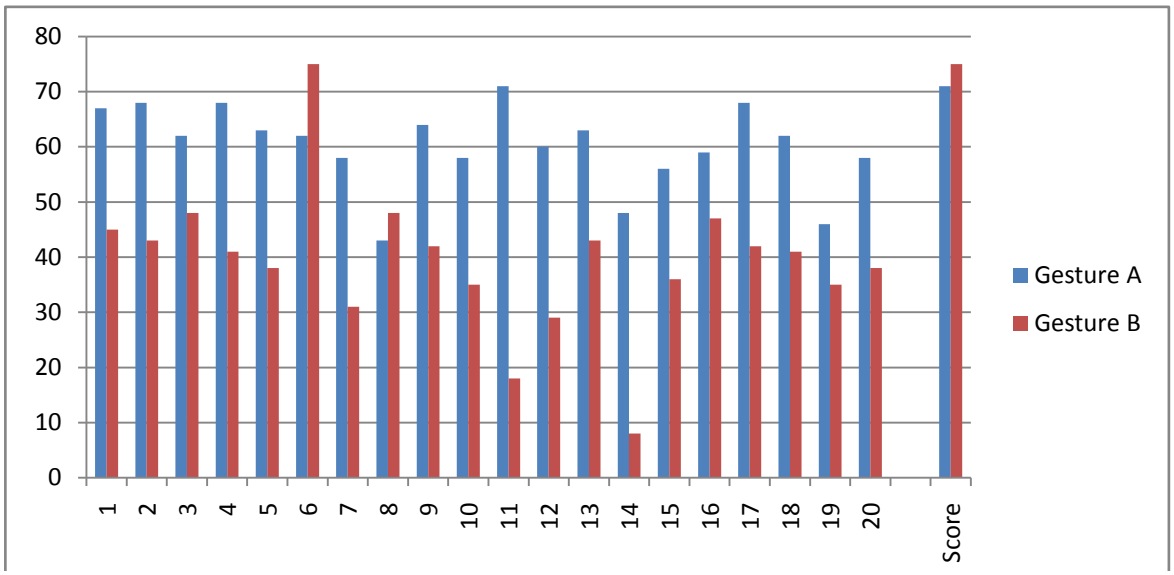


Figure 78: The user's gesture is being compared to gesture A and gesture B. Each gesture has 20 samples. It seems that gesture A is the right one as it has a better overall result. However the gesture will be identified as gesture B as a singles sequence of the sequence set of gesture B has a higher score than any sequence of A.

To avoid such a problem the score of each gesture could be the mean of the n highest DTW scores instead of the highest one.

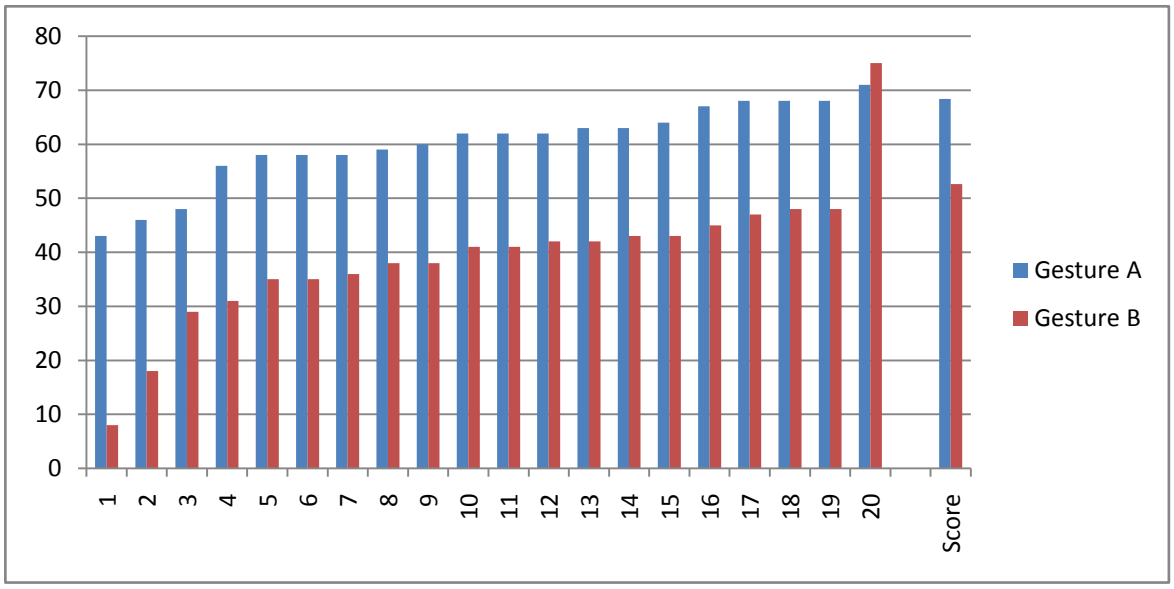


Figure 79: Sequences have been ordered in ascending score manner. The score has been computed has the mean of the 5 sequences with highest score. In this case the system identifies the gesture as gesture A correctly.

14.4 Install it on an AGV

As the product developed in this project was intended to be working on an AGV on of the next logic steps is to actually install it on an AGV.

14.5 Online learning

One way to improve the liability of the system is to increase the model sequences for each gesture. The more sequences there are, the higher will be the score of the correct gesture. Therefore the identification score threshold can be risen reducing false positives.

Gesture sequences can be stored offline, but also online. If a user performs a gesture and the obtained score is over a certain threshold it can be stores as a new sequence of the identified gesture.

The problem is that if the threshold is to low the new sequences of a same gesture will be more and more distant over time. For example imagine sequences are scalars instead of vectors. The model gesture is 10, the threshold to save new sequences is that the difference is equal or less than 1 and the user performs a gesture with score of 9. This sequence will be stored as the difference between 10 and 9 is 1. The user performs again a gesture with a score of 8. This sequence will also be stored as the difference between the performed gesture and the second sequence is 1 even though it has a difference of 2 in respect of the original sequence.

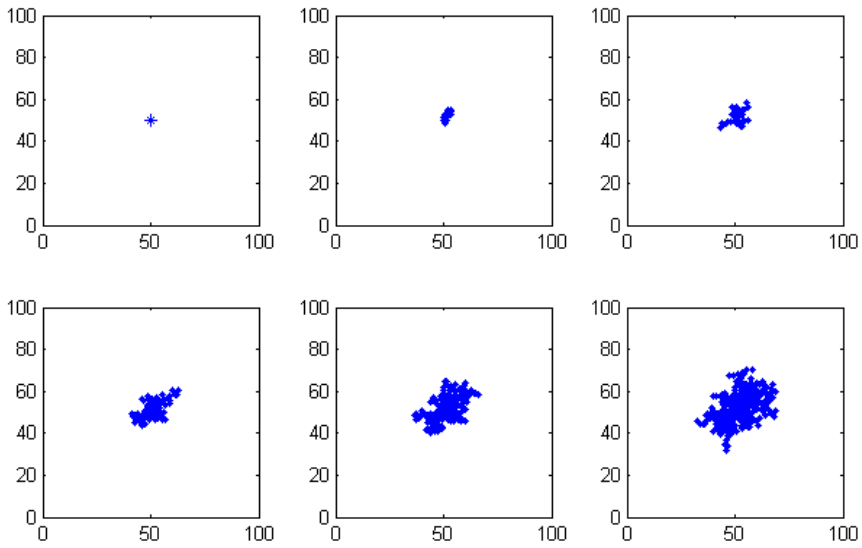


Figure 80: 2-dimensional example. New points will only by added if they have a Euclidean distance lower than 2.5 in respect of an existing point. It shows as new points scatter around the plane.

If the threshold is high sequences will be added slowly and variance will almost not be changed.

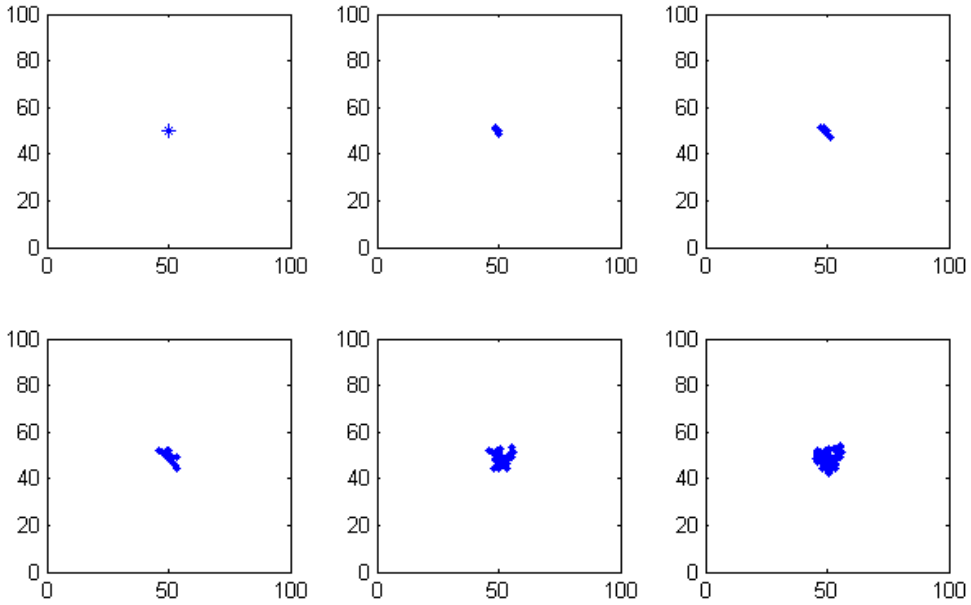


Figure 81: 2-dimensional example. New points will only by added if they have a Euclidean distance lower than 2 in respect of an existing point. It grows slowly.

To have a fast growing database but not scattering new sequences too much a dynamic threshold can be used. This threshold rises in function of the number of stored sequences.

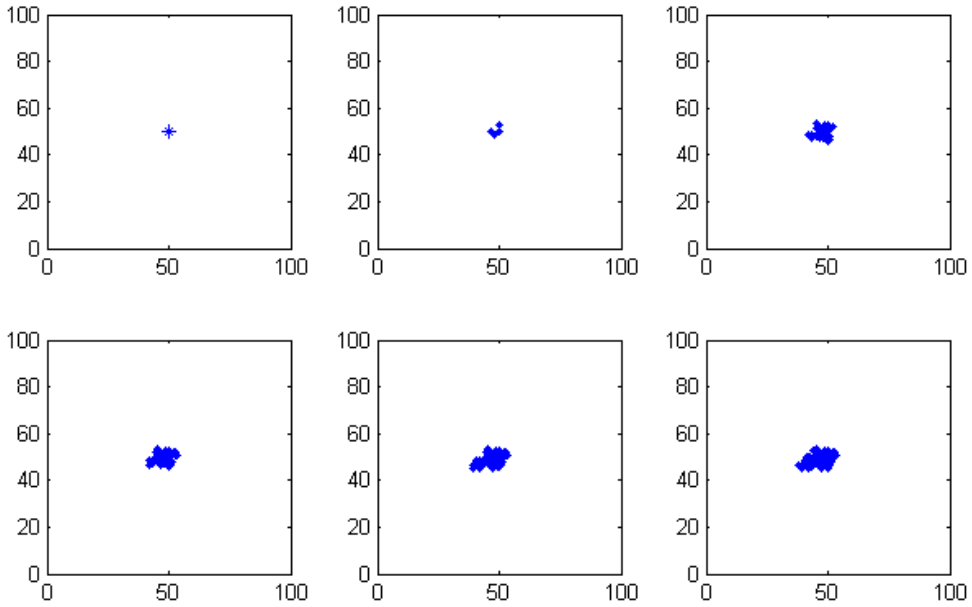


Figure 82: 2-dimensional example. New points will only by added if they have a Euclidean distance lower than $\frac{3}{\text{number of stored sequences} * 0.02}$ in respect of an existing point. It grows fast but maintains its variance low.

14.6 Touchscreen

Finally, as the system is provided by a screen, it could be extended to be a touchscreen. A GUI could be designed in which a technician would be able to see technical information about the AGV and its status, such as errors and would also be able to change parameters of the AGV or the gesture recognition system.

CHAPTER 15: PROJECT COST

This chapter will try to resume the development cost of the project. The cost can be divided in different categories.

15.1 Workforce cost (fixed)

All the tasks for the development of this project have been done by an engineer, the author. Therefore the workforce cost is the same for all tasks: 10,93€/h. A total of 504 hours have been spent, thus increasing the overall cost to 5508,72€. Each task and its duration can be seen in figureFigure 83:

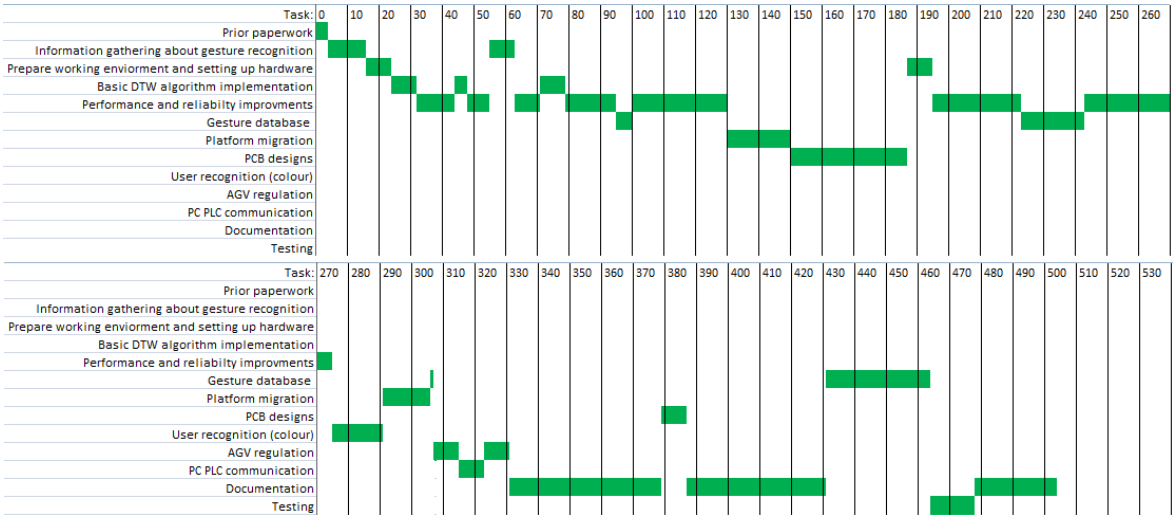


Figure 83: Performed tasks and their durations

15.2 Licenses costs (fixed)

The development software need licenses to work properly. The first attempts for gesture recognition were programmed on the MATLAB suite. The code has been later migrated to C# and XAML using Visual Studio. To work with Kinect for XBOX One the Kinect SDK is needed.

License	Price
Standard MATLAB license	2.000,00 €
Visual Studio Professional 2013	647,00 €
Kinect SDK v2.0	0.00 €
Total cost:	2647,00 €

Table 4: Licenses costs (fixed)

15.3 License costs (variable)

Each AGV will have an Intel NUC PC. This PC runs the operating system Windows 8.1 Pro. It may be changed for a Windows 8.1 Embedded Standard.

License	Price
Windows 8.1 Pro	279,00 €
Total cost:	279,00 €

Table 5: Licenses costs (variable)

15.4 Hardware cost (variable)

As stated before each AGV carries an Intel NUC PC. This PC has no hard drive nor memory. A 60GB SSD drive has been installed in the prototype. This could be reduced to a 30GB drive in production models. The prototype has an 8GB RAM memory, which could be downsized to 2GB.

An 9 inch touchscreen will be mounted on the AGV. It also need a HDMI cable.

To communicate the PC with the PLC a USB to Serial adapter is used.

Finally, the price of the PCBs includes manufacturing, component prices and soldering.

Hardware	Price
Intel NUC D54250WYK	313,22 €
Crucial DDR3 1600 PC3-12800 8GB CL11 SO-DIMM	42,15 €
Kingston SSDNow mS200 60GB mSATA	39,67 €
HDMI cable	2,27 €
9 inch HDMI screen	108,55 €
USB-Serial Adapter	13,50 €
Power supply PCB	65,50 €
Power control PCB	82,70 €
Kinect for XBox ONE	149,99 €
Kinect for Windows adapter	49,99 €
Total cost:	867,54 €

Table 6: Hardware cost

15.5 Total cost

The total cost is divided in fixed (onetime expenses) and variable (expenses for every unit) costs.

Fixed Cost	Price
Workforce	5508,72€
Licenses	597,56 €
Total cost:	6106.28 €

Table 7: Total fixed costs

Variable Cost	Price
Licenses	279,00 €
Hardware	867,54 €
Total cost:	1146,54 €

Table 8: Total variable costs

CHAPTER 16: GLOSSARY

AGV: Automatic guided vehicle. An autonomous navigating robot used especially in warehouses and factories to transport goods from one place to another automatically.

Avatar: The avatar is a graphical representation of a character shown on a screen. [57]

CAN: Controller Area Network is a communication protocol designed by Robert Bosch GmbH designed especially for vehicles. [58]

Confusion matrix: Used in the field of machine learning a confusion matrix (also known as contingency table or error matrix) is used to visualize the performance of a supervised learning algorithm. [59]

DDTW: Derivative Dynamic Warping. An enhanced DTW algorithm that uses derivatives.

DTW: Dynamic Time Warping. An algorithm used to synchronize and compare signals.

GEMMA Guide: The GEMMA Guide is a methodology to study all possible states of an automatism and design its security systems. [60]

GFLOPS: Thousand millions floating point operations per second. Measure for computer power.

GRAFCET: The GRAFCET is graphical representation of the successive behaviours a logical system has defined by its inputs and outputs. [61]

GUI: Graphical user interface. It is a type of interface used in screen displaying text and images with which the user can interact through a mouse, keyboard or touchscreen.

HRI: Human Robot Interaction. The study of interactions between humans and robots.

HSL and HSV: Hue, Saturation and Lightness or Value. It is a model of colour.

Interaction: An action that occurs as two or more objects have an effect upon one another. [62] In engineering it is usually focused on the interaction between user and machinery.

Kalman filter: The Kalman filter is an algorithm that uses measurements observed over time with statistical noise to produce estimations of an unknown variable. It may be used as a filter. [63]

Kinect Sensor: A 3 dimensional camera developed by Microsoft.

Likert scale: The Likert Scale is a psychometric scale involved in sociologic research that employees questionnaires. [64]

MDDTW: MultiDimensional Dynamic Time Warping. An enhanced DTW algorithm that uses multiple dimensions.

PCB: Printed Circuit Boards mechanically supports and electrically connects electronic components on a copper sheet. [65]

PLC: Programmable logic controller. A robust controller used mainly in industrial automation.

RGB: Red, Green and Blue. A colour model.

ROS: Robot operating system. A middleware to communicate and control different hardware of a robot.

RS232:A standard for serial communication transmission of data. [66]

SCADA: Supervisory control and data acquisition.

Technician: A special user that has the sufficient knowledge of a system to maintain it and fix it.

User: A person who uses a product/machinery.

Virtual Assistant: A virtual person who helps the users to accomplish its tasks. It is usually represented by an avatar.

WDTW: Windowed Dynamic Time Warping. An enhanced DTW algorithm which reduces the possible time deviation.

CHAPTER 17: REFERENCES

[1] Hidden Markov Model for Gesture Recognition Jie Yang; YangshengXu, Carnegie Mellon University, Robotics Institute, May, 1994

[2] Spotting Recognition of Human Gestures from Motion Images, K.Takahashi; S.Seki; R.Oka, The Inst. of ElectromCS, Information and Comm, vol.36, No.7, pp.28-35, 1993.

[3] Microsoft Kinect v2 Driver Released *http://www.ros.org/news/2014/09/microsoft-kinect-v2-driver-released.html*

[4] Neural Networks with Hidden Markov Models in Skeleton-Based Gesture Recognition, Hai-Son Le; Ngoc-Quan Pham; Duc-Dung Nguyen

[5] Multi-Dimensional Dynamic Time Warping for Gesture Recognition, G.A. ten Holt; M.J.T. Reinders; E.A. Hendriks, Thirteenth annual conference of the Advanced School for Computing and Imaging, 2007

[6] Exact indexing of dynamic time warping, Eamonn Keogh; Chotirat Ann Ratanamahatana, University of California–Riverside, Computer Science and Engineering Department, Riverside, USA, 2003

[7] Gesture recognition using skeleton data with weighted dynamic time warping, Sait Celebi [et. al.], Istanbul Sehir Univeristy, Istanbul.

[8] Robust Fisher Discrimination Analysis, Kim, S.-J.; Magnami, A.; Boyd, S.P., Neural Information Processing Systems, 2005.

[9] Iridis-pi: a low-cost, compact demonstration cluster, Simon J. Cox [et. al.], 2013

[10] Dynamic Programming Algorithm Optimization for Spoken Word Recognition, Hiroaki Sakoe; Seibi Chiba, IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, VOL. ASSP-26, NO. 1, 1978

[11] Learning DTW Global Constraint for Time Series Classification, Vit Niennattrakul; Chotirat Ann Ratanamahatana, Department of Computer Engineering, Chulalongkorn University, 2009.

[12] Minimum Prediction Residual Principle Applied to Speech Recognition, Fumitada Itakura, 1975

[13] Faster Retrieval with a Two-Pass Dynamic-Time-Warping Lower Bound,Lemire, D., Pattern Recognition 42, 2009

[14] Real-Time Human Pose Recognition in Parts from Single Depth Images, Jamie Shotton [et. al.], Microsoft Research Cambridge & Xbox Incubation

[15] HSL and HSV, Wikipedia http://en.wikipedia.org/wiki/HSL_and_HSV

[16] GS-2744B Guide Sensor INSTRUCTIONS, MACOME, 2011

[17] Forklifts vs. Automated Guided Vehicles, Adaptalift, http://www.aalhysterforklifts.com.au/index.php/about/blog-post/forklifts_vs._automated_guided_vehicles

[18] AiTech products, AiTech http://www.aitech-robotics.com/index.php/products/automated-guided-vehicle-agv

[19] AGV Advantages over Manual Transport Systems, Corecon, http://www.coreconagvs.com/sales/return_on_investment.php

[20] Introduction of OK Knaspack Pull AGV AK-AGV500, OKAGV http://www.wxagv.com/Back-Pull_AGV14661933.html

[21]Minimum prediction residual principle applied to speech recognition, Fumitada Itakura, IEEE Transactions on Acoustics, Speech, and Signal Processing, 1975.

[22] Automated guided vehicle, Wikipedia http://en.wikipedia.org/wiki/Automated_guided_vehicle

[23] About AGVs, HMI http://www.mhi.org/agvs

[24] AGVs Explained, NDC Automation http://www.ndcautomation.com/?page_id=272

[25] Automated Guided Vehicle Law & Legal Definition, USLegal http://definitions.uslegal.com/a/automated-guided-vehicle/

[26] Automated Guided Vehicle System (AGVS), BusinessDictionary.com http://www.businessdictionary.com/definition/Automated-Guided-Vehicle-System-AGVS.html

[27] On-line Training Program, What is an AGV?, MHI http://www.mhi.org/downloads/industrygroups/agvs/elessons/what-is-an-agv.pdf

[28] Kivnon Logística http://www.kivnon.com/

[29] Human-Automation Interaction, Thomas B. Sheridan; Raja Parasuraman, 2006

[30] Beyond Embodiment and Social Presence: Preferences for Virtual Assistant Gender and Clothing Style, Jeunese Adrienne Payne [et. al.], 2011

[31] Gendering the Machine: Preffered Vritual Assistan Gender and Realism in Self-service, Jeunese Adrienne Payne [et. al.], 2012

[32] The Influence of the Avatar on OnlinePerceptions of Anthropomorphism,Androgyny, Credibility, Homophily,and Attraction, Kristine L. Nowk; Christian Rau, 2006

[33] Too real for comfort? Uncanny responses to computer generated faces, Karl F. MacDorman [et. al.], 2009

[34] 不気味の谷現象 (Bukimi no tanigenshou), Masashiro Mori, 1970

[35] Is happy better than sad even if they are bothnon-adaptive? Effects of emotional expressions of talking–head interface e-agents, Gong, L., 2007.

[36] Derivative Dynamic Time Warping, Eamonn J. Keogh; Michael J. Pazzani, 2001

[37] The Kalman Filter, Juan Andrade-Cetto, 2002

[38] Design and evaluation of an avatar-based help system for home automation scenarios, Pere Ponsa [et. al.], 2012

[39] Natural User Interfaces Are Not Natural, Don Norman, 2010

[40] Usability of Vision-Based Interfaces, Cristina Manresa-Yee; Esperança Amengual; Pere Ponsa, 2013

[41] Color Appearance Models: CIECAM02 and Beyond, Mark Fairchild, 2004

[42] GEMMA, Universidad de Oviedo
http://isa.uniovi.es/docencia/iea/teoria/gemma_resumen.pdf

[43] Modos de marcha y parada La guía GEMMA, Victor M. Gonzáles Suárez, Universidad de Oviedo
http://isa.uniovi.es/~vsuarez/Download/GemmaTelemecanique.PDF

[44] Introducción al modelado GRAFCET, Univesidad Politénica de Madrid http://www.elai.upm.es/moodle/pluginfile.php/1171/mod_resource/content/0/GrafcetAmpliacion.pdf

[45] Resumen sobre GRAFCET, Universidad de Oviedo http://isa.uniovi.es/docencia/iea/teoria/grafcet_resumen.pdf

[46] Understanding the Advantages and Disadvantages of Linear Regulators, Steven Keeping, 2012, DigiKey http://www.digikey.com/es/articles/techzone/2012/may/understanding-the-advantages-and-disadvantages-of-linear-regulators

[47] LM2676 SIMPLE SWITCHER® High Efficiency 3A Step-Down Voltage Regulator, Texas Instruments, 2013 http://www.ti.com/lit/ds/symlink/lm2676.pdf

[48] Electronic housing - ME MAX 22,5 3-3 KMGY, Phoenix Contact https://www.phoenixcontact.com/online/portal/us?uri=pxc-oc-itemdetail:pid=2713939&library=usen&tab=1

[49] Gesture Recognition - Algorithms, Wikipedia https://en.wikipedia.org/wiki/Gesture_recognition#Algorithms

[50] Two-stage Hidden Markov Model in Gesture Recognition for Human Robot Interaction, Nhan Nguyen-Duc-Thanh; Sunyoung Lee; Donghan Kim, Kyung Hee University, South Korea, 2012

[51] Kinect-aided Robust Gesture Recognition for Human-Robot Interaction with Application to Quadrocopter Control, Oliver Brand; Max Lungarella; Davide Scaramuzza, University of Zurich, 2013

[52] Using Human Gestures and Generic Skills to Instruct a Mobile Robot Arm in a Feeder Filling Scenario, Mikkel Rath Pedersen; Carsten Høilund; Volker Kruger, Copenhagen, 2012

[53] People Following Automated Guided Vehicles - Research and Application, Loathar Schulze; Sebastian Behling; Stefan Buhrs, Hong Kong, 2009

[54] Literature Review of Mobile Robots for Manufacturing, Michael Shneier; Roger Bostelman, May 2015

[55] Likert scale https://en.wikipedia.org/wiki/Likert_scale

[56] CAN vs. RS485, IXXAT http://www.ixxat.com/download/artikel_20105_can-vs-rs485_e.pdf

[57] Avatar (computing), Wikipedia https://en.wikipedia.org/wiki/Avatar_(computing)

[58] CAN bus, Wikipedia https://en.wikipedia.org/wiki/CAN_bus

[59] Confusion matrix, Wikipedia https://en.wikipedia.org/wiki/Confusion_matrix

[60] Guía de studio de los modos de marchas y paradas, Wikipedia https://es.wikipedia.org/wiki/Gu%C3%ADa_de_estudio_de_los_modos_de_marchas_y_paradas

[61] GRAFCET, Wikipedia https://es.wikipedia.org/wiki/GRAFCET

[62] Interaction, Wikipedia https://en.wikipedia.org/wiki/Interaction

[63] Kalman Filter, Wikipedia https://en.wikipedia.org/wiki/Kalman_filter

[64] Likert Scale, Wikipedia https://en.wikipedia.org/wiki/Likert_scale

[65] Printed circuit board, Wikipedia https://en.wikipedia.org/wiki/Printed_circuit_board

[66] RS-232, Wikipedia https://en.wikipedia.org/wiki/RS-232

[67] Skeletal Joint Smoothing White Paper: Why We Need Joint Filtering, Microsoft https://msdn.microsoft.com/en-us/library/jj131429.aspx

CHAPTER 18: ADDITIONAL BIBLIOGRAPHY

Introduction to Random Signals and Applied Kalman Filtering with MATLAB Exercises, Robert Grover Brown; Patrick Y.C. Hwang, 1997

Applied Optimal Estimation, Arthur Gelb, 1974

Human-Agent and Human-Robot Interaction Theory: Similarities to and Differences from Human-Human Interaction, Nicole C. Krämer; Astrid von der Pütten; Sabrina Eimler, 2012

AGV K11BGMP1-400 Manual de servicio, Kivnon Logística, 2015

Kinect for Windows v2 Windows Runtime API Reference, Microsoft https://msdn.microsoft.com/en-us/library/dn758675.aspx

.NET Framework Class Library, Microsoft https://msdn.microsoft.com/en-us/library/gg145045(v=vs.110).aspx

Windows Presentation Foundation, Microsoft https://msdn.microsoft.com/en-us/library/ms754130(v=vs.100).aspx

SIMATIC S7-1200 Programmable controller System Manual, Siemens, 2012

RSLogix 500 and RSLogix Micro Getting Results Guide, Rockwell Automation, 2011

CANopen high-level protocol for CAN-bus, H. Boterenbrood, 2000

CHAPTER 19: ANNEX

19.1 Formulae to obtain angles of right arm:

$$L_{NS} = J_S - J_N \quad (18)$$

$$L_{SE} = J_E - J_S \quad (19)$$

$$L_{EW} = J_W - J_E \quad (20)$$

$$L_{WT} = J_T - J_W \quad (21)$$

$$L_{WH} = J_H - J_W \quad (22)$$

$$\beta = atan2(L_{NS}(z), L_{NS}(x)) \quad (23)$$

$$R_{NS} = \begin{bmatrix} cos(\beta) & 0 & -sin(\beta) \\ 0 & 1 & 0 \\ sin(\beta) & 0 & cos(\beta) \end{bmatrix} \quad (24)$$

$$L_{SE(NS)} = R_{NS} L_{SE} \quad (25)$$

$$R_{SE} = \left[\frac{L_{SE}}{\|L_{SE}\|} \quad \frac{L_{SE} \times L_{EW}}{\|L_{SE} \times L_{EW}\|} \times \frac{L_{SE}}{\|L_{SE}\|} \quad \frac{L_{SE} \times L_{EW}}{\|L_{SE} \times L_{EW}\|} \right] \quad (26)$$

$$R_{SE}' = \left[\frac{L_{SE}}{\|L_{SE}\|} \quad \frac{L_{SE} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}}{\left\| L_{SE} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right\|} \times \frac{L_{SE}}{\|L_{SE}\|} \quad \frac{L_{SE} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}}{\left\| L_{SE} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right\|} \right] \quad (27)$$

$$Y_{R_{SE}(R_{SE}')} = R_{SE}' R_{SE}(y\ column) \quad (28)$$

$$L_{EW(SE)} = R_{SE} L_{EW} \quad (29)$$

$$R_{EW} = \left[\frac{L_{EW}}{\|L_{EW}\|} \quad \frac{L_{WT} \times L_{EW}}{\|L_{WT} \times L_{EW}\|} \quad \frac{L_{EW}}{\|L_{EW}\|} \times \frac{L_{WT} \times L_{EW}}{\|L_{WT} \times L_{EW}\|} \right] \quad (30)$$

$$R_{EW}' = \left[\frac{L_{EW}}{\|L_{EW}\|} \quad \frac{L_{EW}}{\|L_{EW}\|} \times \frac{L_{SE} \times L_{EW}}{\|L_{SE} \times L_{EW}\|} \quad \frac{L_{SE} \times L_{EW}}{\|L_{SE} \times L_{EW}\|} \right] \quad (31)$$

$$Z_{R_{EW}(R_{EW}')} = R_{EW}' R_{EW}(z\ column) \quad (32)$$

$$UpperArm\ Yaw = Atan2\left(L_{SE(NS)}(z), L_{SE(NS)}(x)\right) \quad (33)$$

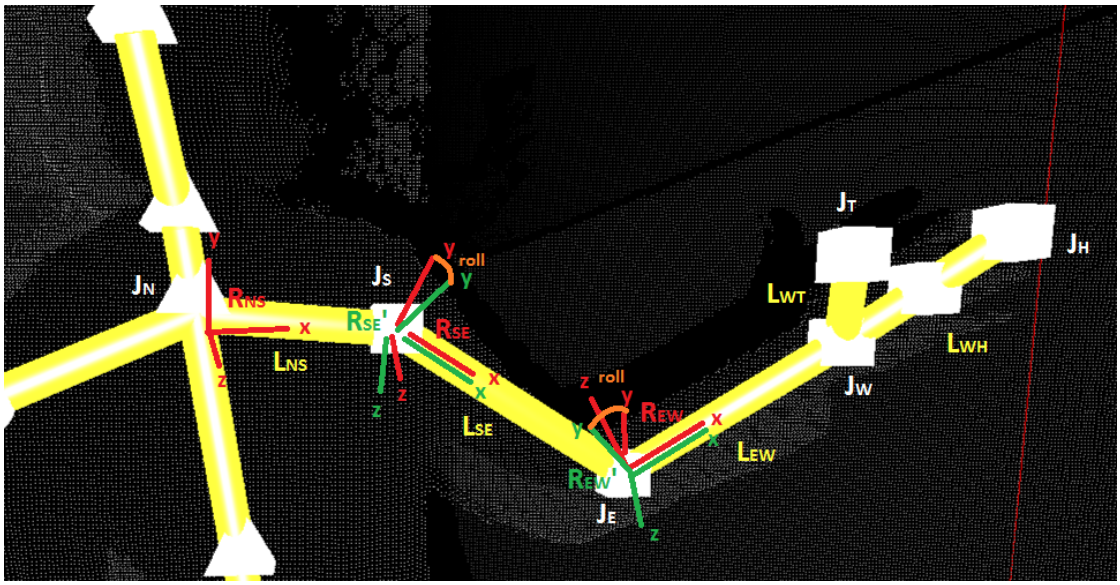
$$Upper\ Arm\ Pitch = -Atan2\left(L_{SE(NS)}(y), \sqrt{\left(L_{SE(NS)}(x)^2 + L_{SE(NS)}(z)^2\right)}\right) \quad (34)$$

$$Upper\ Arm\ Roll = Atan2\left(Y_{R_{SE}(R_{SE}')}(z), Y_{R_{SE}(R_{SE}')}(y)\right) \quad (35)$$

$$Forearm\ Pitch = Atan2\left(L_{EW(SE)}(y), L_{EW(SE)}(x)\right) \quad (36)$$

$$Forearm\ Roll = Atan2\left(Z_{R_{EW}(R_{EW}')}(z), Z_{R_{EW}(R_{EW}')}(y)\right) \quad (37)$$

$$Hand\ Pitch = Atan2\left(L_{WH(EW)}(y), L_{WH(EW)}(x)\right) \quad (38)$$



19.2 Formulae to obtain weights of Weighted Dynamic Time Warping:

First the gesture to be analysed has to be performed by a trained user. Once all features have been obtained the contribution of each feature is calculated for the gesture. The formula is:

$$D_f^g = \sum_{n=2}^{N} \left| S_{f,n}^g - S_{f,n-1}^g \right| \quad (39)$$

where g is the gesture index, f the feature index and n the frame.

The next step consists in a filtering and threshold (T_{min}, T_{max}) operation:

$$D_f^g = \begin{cases} D_a & if\ 0 \le D_f^g < T_{min} \\ \dfrac{D_f^g - T_{min}}{T_{max} - T_{min}} (D_b - D_a) + D_a & if\ T_{min} \le D_f^g < T_{max} \\ D_b & otherwise \end{cases} \quad (40)$$

where D_a and D_b are threshold values.

The weights are then calculated in function of D_f^g and β. β is a parameter to be optimized later.

$$w_f^g = \frac{1 - e^{-\beta D_f^g}}{\sum_k \left(1 - e^{-\beta D_k^g}\right)} \quad (41)$$

This weights are introduced in the DTW matrix cost function while comparing a gesture b to the model a as:

$$DTWmatrix_{cost}(x,y) = \sum_{f=1}^{F} w_f^a \left| S_{f,x}^a - S_{f,y}^b \right| \quad (42)$$

Finally the optimal β* parameter must be found maximizing the expression:

$$\beta^* = \arg\max_{\beta} \frac{D_B}{D_W} \quad (43)$$

D_B is the between-class dissimilarity. It corresponds to the formula:

$$D_B = \sum_{m} \sum_{\substack{n \\ n \neq m}} D_{mn} \quad (44)$$

where D_{mn} is the average weighted DTW distance of all samples of gesture class m and gesture class n.

D_B is the within-class dissimilarity. It is the average DTW distance of all samples of a class with respect to each other.

19.3 Formulae to obtain HSV from RGB:

$$red' = \frac{red}{255} \quad (45)$$

$$green' = \frac{green}{255} \quad (46)$$

$$blue' = \frac{blue}{255} \quad (47)$$

$$C_{max} = \max(red', green', blue') \quad (48)$$

$$C_{min} = \min(red', green', blue') \quad (49)$$

$$C_{\Delta} = C_{max} - C_{min} \quad (50)$$

$$k = \begin{cases} 0 & , green' => blue' \\ 6 & , green' < blue' \end{cases} \quad (51)$$

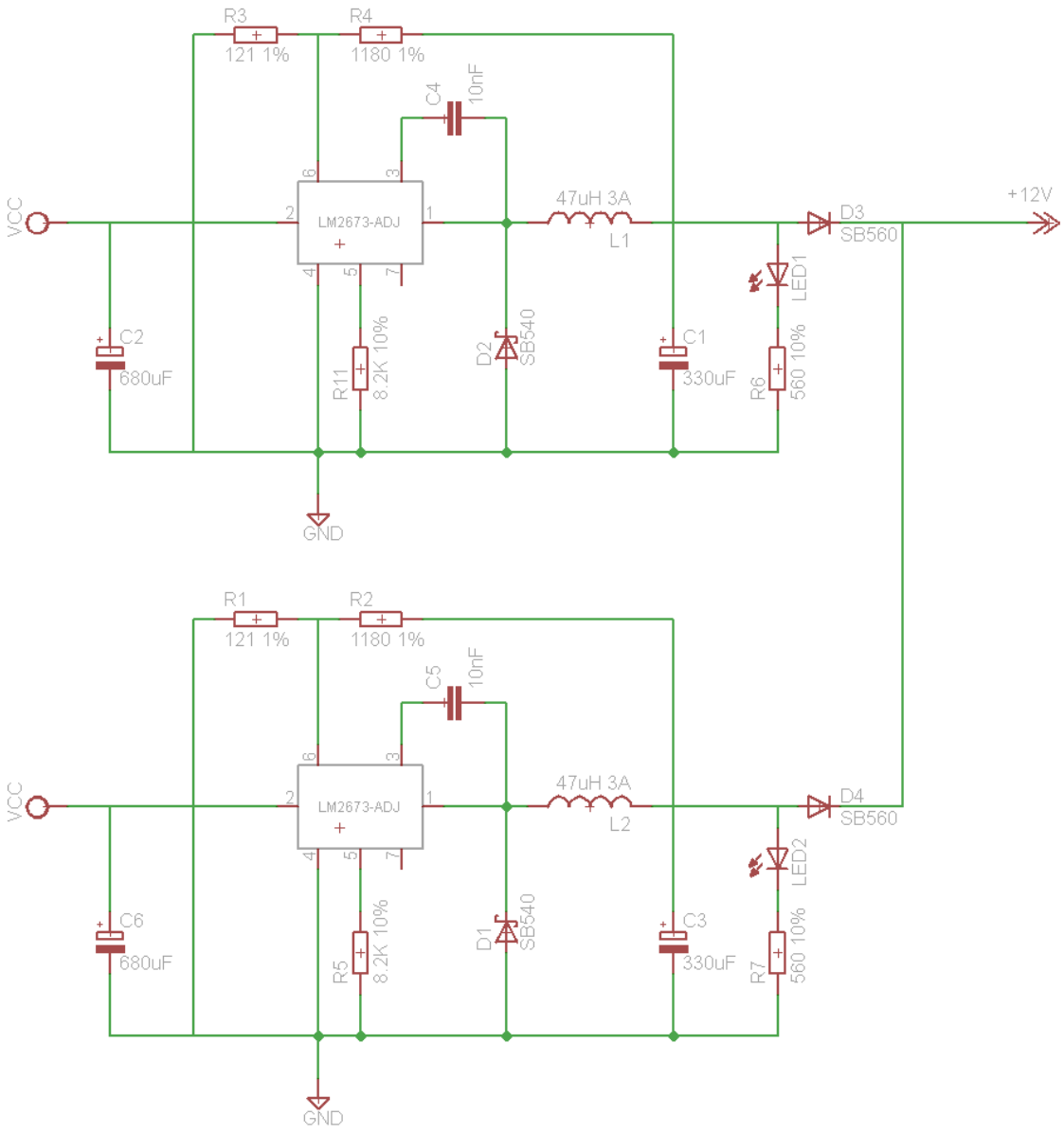
$$Hue = \begin{cases} 0 & , C_{max} = C_{min} \\ 60 * \left(k + \frac{green' - blue'}{C_{\Delta}} \right) & , C_{max} = red' \\ 60 * \left(\frac{green' - blue'}{C_{\Delta}} \right) & , C_{max} = green' \\ 60 * \left(\frac{green' - blue'}{C_{\Delta}} \right) & , C_{max} = blue' \end{cases} \quad (52)$$

$$Saturation = \begin{cases} 0 & , C_{max} = 0 \\ \frac{C_{\Delta}}{C_{max}} & , C_{max} \neq 0 \end{cases} \quad (53)$$

$$Value = C_{max} \quad (54)$$

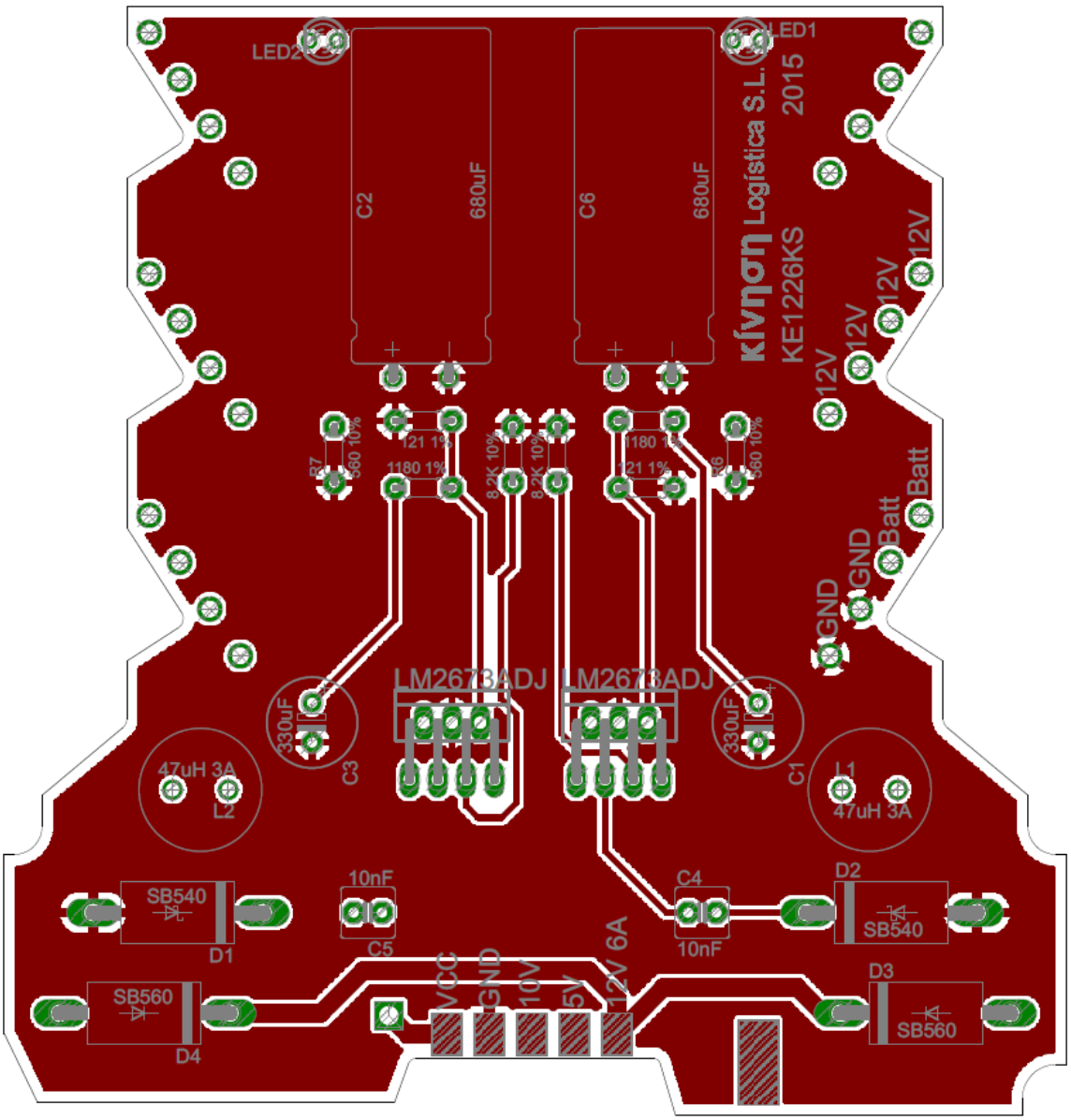
19.4 Power supply PCB

19.4.1 Schematic

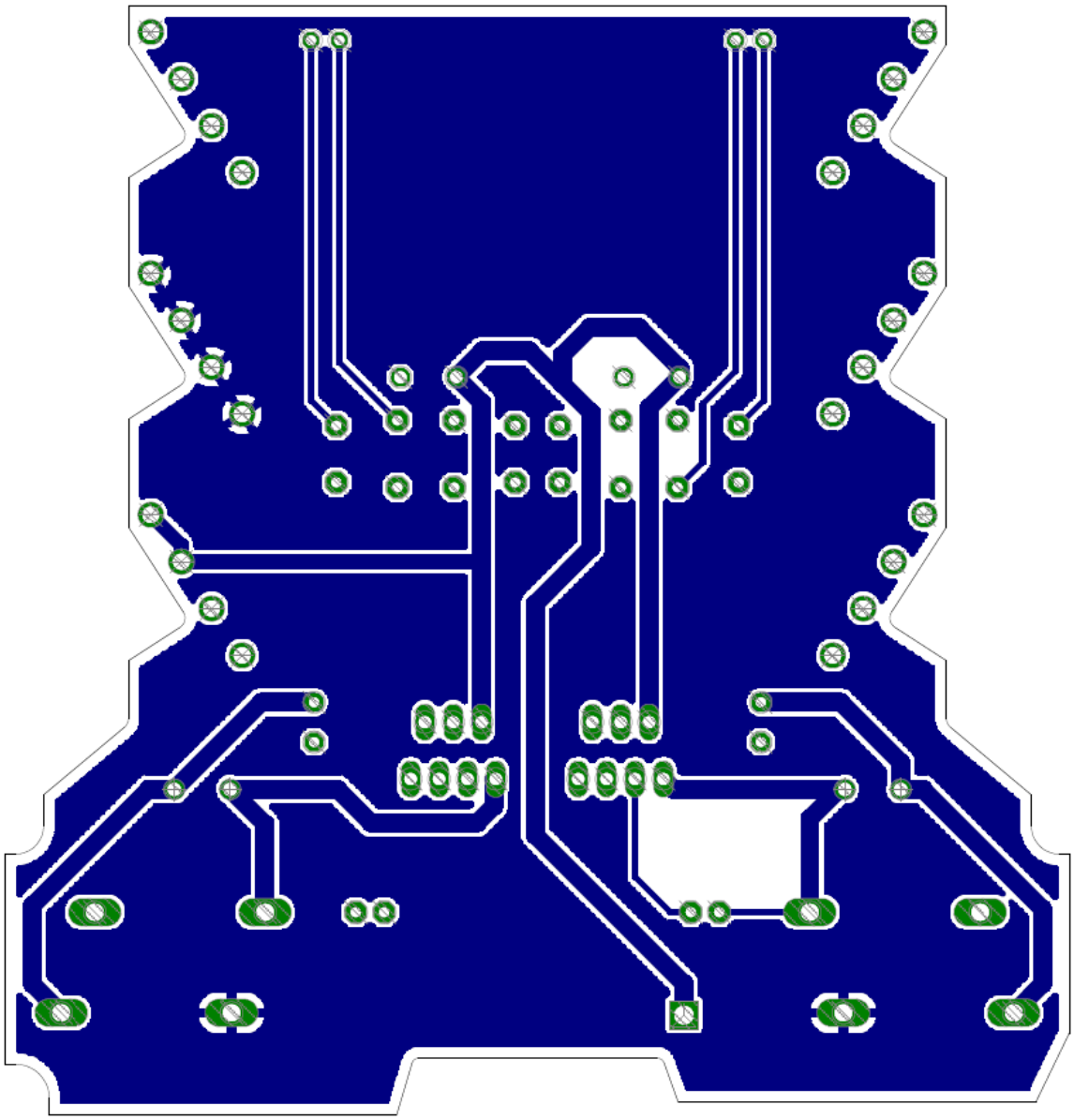


19.4.2 PCB Layout

Top

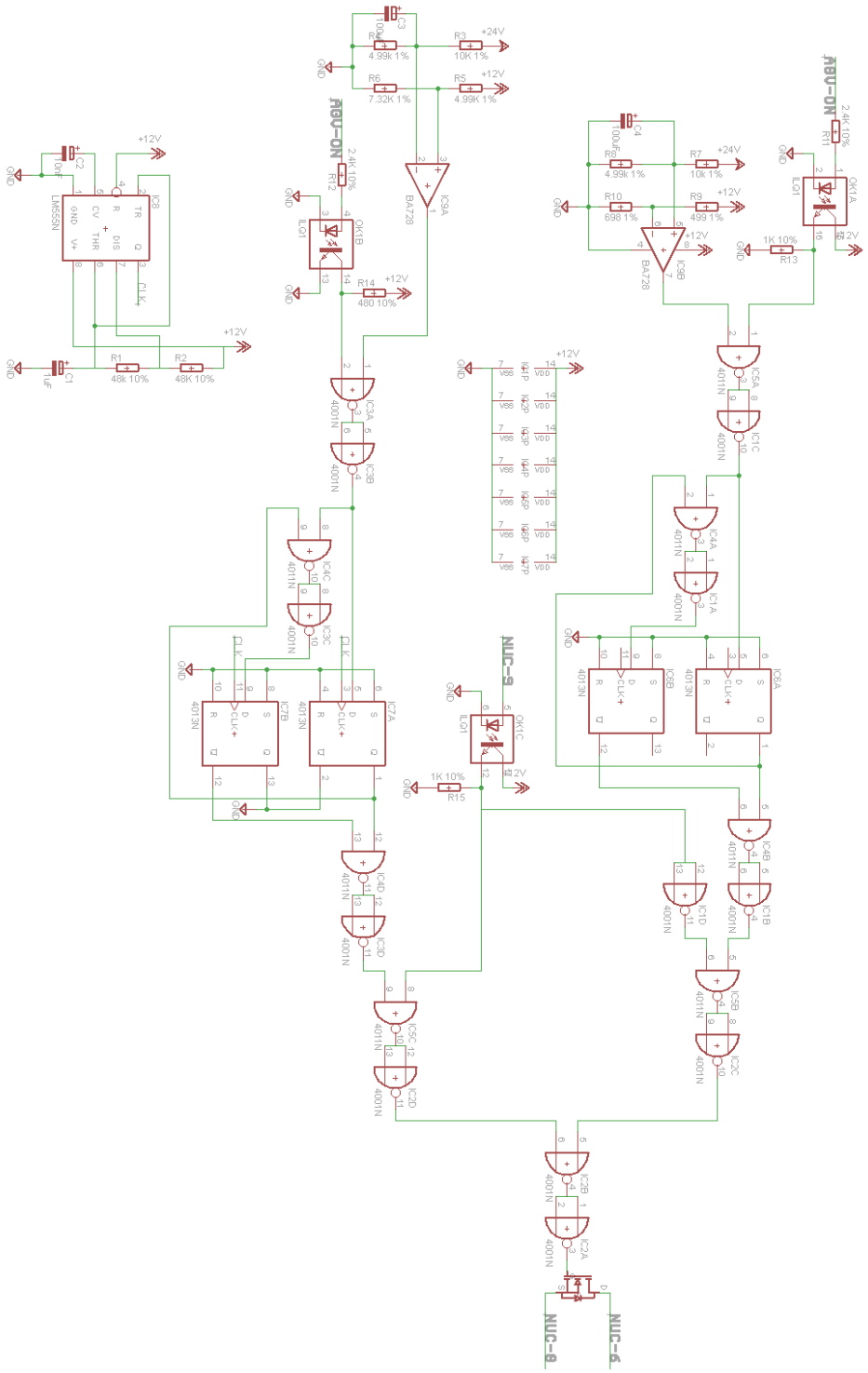


Bottom



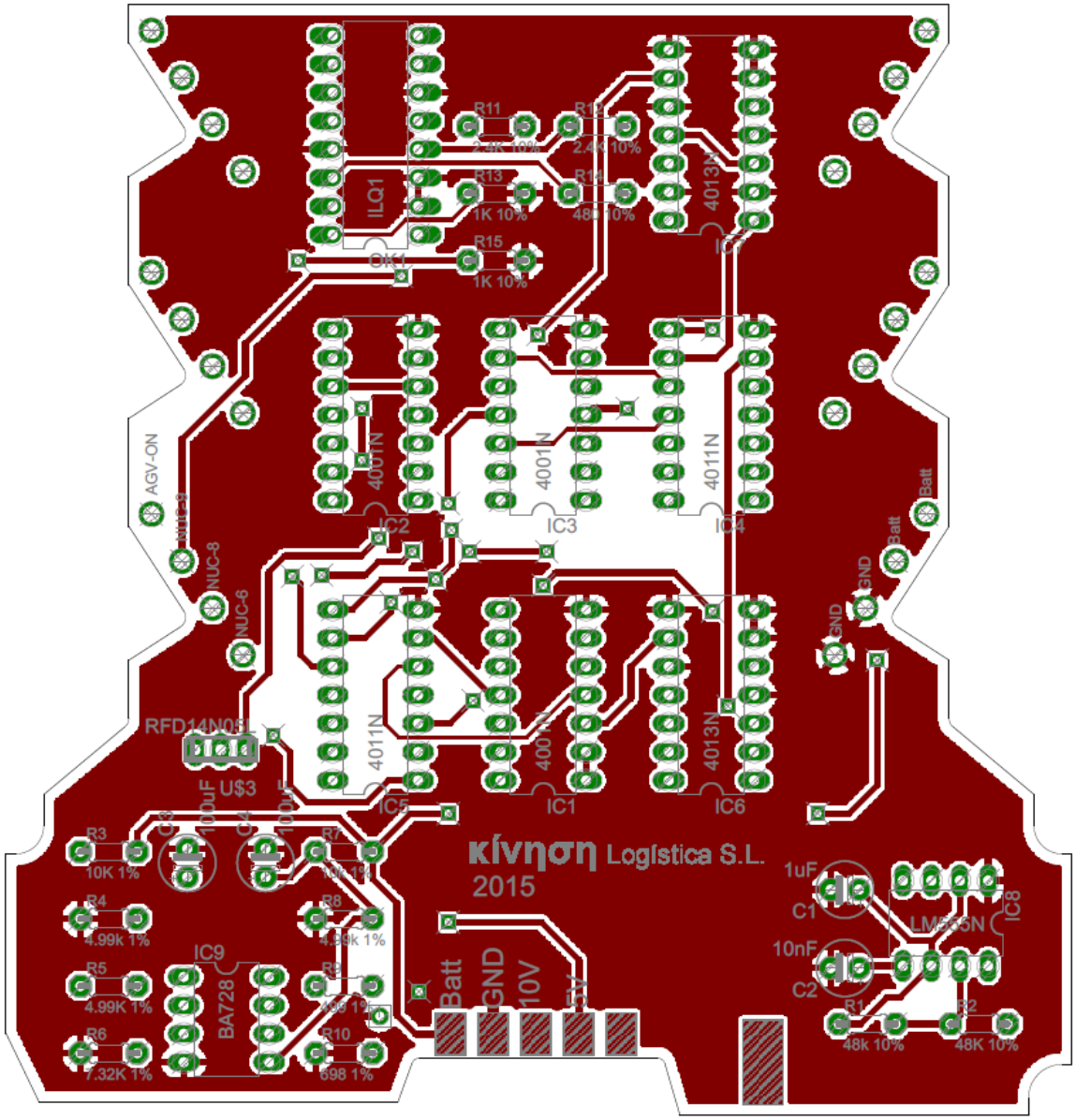
19.5 Power controller PCB

19.5.1 Schematic

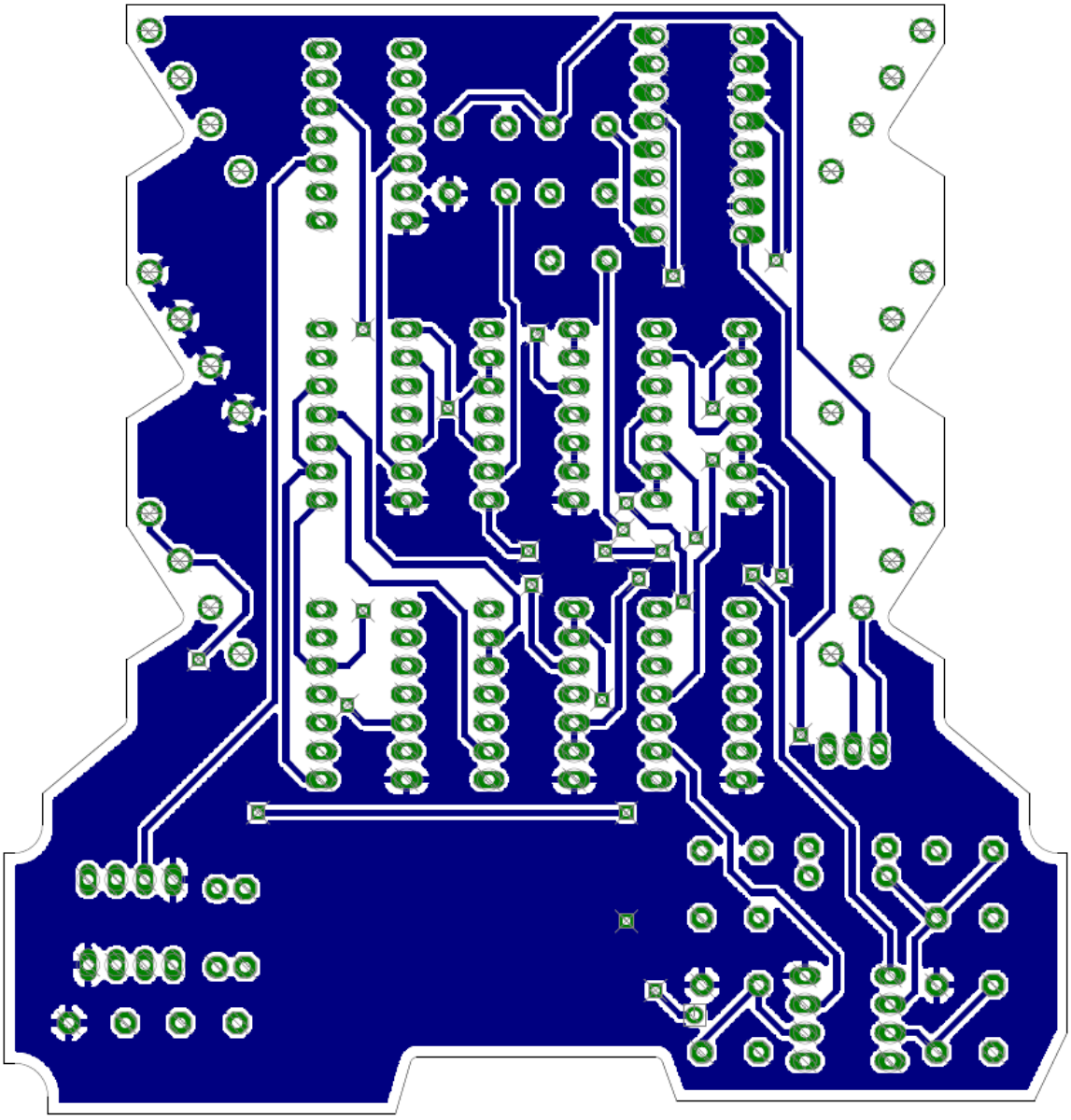


19.5.2 PCB Layout

Top



Bottom



19.6 Main program's full code with comments

19.6.1 MainWindows.xaml

```
<Windowx:Class="KinectAGVNavigator.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Body Basics"
Height="1000"Width="1840"
Loaded="MainWindow_Loaded"
Closing="MainWindow_Closing">
<Window.Resources>
<SolidColorBrushx:Key="MediumGreyBrush"Color="#ff6e6e6e"/>
<SolidColorBrushx:Key="KinectPurpleBrush"Color="#ff52318f"/>
<SolidColorBrushx:Key="KinectBlueBrush"Color="#ff00BCF2"/>
</Window.Resources>
<GridMargin="10 0 10 0">

<Grid.RowDefinitions>
<RowDefinitionHeight="Auto"/>
<RowDefinitionHeight="*"/>
<RowDefinitionHeight="Auto"/>
</Grid.RowDefinitions>

<ViewboxGrid.Row="1"HorizontalAlignment="Center">
<ImageSource="{Binding ImageSource}"Stretch="UniformToFill"/>
</Viewbox>

<StatusBarGrid.Row="2"HorizontalAlignment="Stretch"Name="statusBar"VerticalAlignment="Botto
m"Background="White"Foreground="{StaticResource MediumGreyBrush}"Grid.ColumnSpan="2">
<StatusBarItemContent="{Binding StatusText}"/>
</StatusBar>
<TextBlockName="RecordingAnnouncment"HorizontalAlignment="Left"Height="598"Margin="214,
0,0,0"Grid.Row="1"TextWrapping="Wrap"Text="R"VerticalAlignment="Top"Width="755"Foregroun
d="Red"FontSize="400"Visibility="Collapsed"Grid.Column="1"/>
<ListViewName="GestureRecognitionLevel"HorizontalAlignment="Left"Height="896"Margin="120,8,
0,0"Grid.Row="1"Width="214"VerticalAlignment="Top"BorderBrush="{x:Null}"Focusable="False">
<ListView.ItemContainerStyle>
<StyleTargetType="ListViewItem">
<SetterProperty="Height"Value="50"/>
</Style>
</ListView.ItemContainerStyle>
</ListView>
<ListViewScrollViewer.HorizontalScrollBarVisibility="Disabled"Name="GestureRecognitionNames"H
orizontalAlignment="Left"Height="900"Margin="15,8,0,0"Grid.Row="1"Width="107"VerticalAlignm
ent="Top"FlowDirection="RightToLeft"BorderThickness="0,1,1,1"BorderBrush="{x:Null}"Focusable
="False"IsSynchronizedWithCurrentItem="True">
<ListView.ItemContainerStyle>
<StyleTargetType="ListViewItem">
<SetterProperty="Height"Value="50"/>
```

```
</Style>
</ListView.ItemContainerStyle>
</ListView>
<RectangleName="ColorBox"Fill="#FFF4F4F5"HorizontalAlignment="Right"Height="100"Grid.Row
="1"Stroke="Black"VerticalAlignment="Bottom"Width="100"RenderTransformOrigin="0.64,0.46"/>
<TextBoxName="SerialPortText"HorizontalAlignment="Left"Height="35"Margin="0,0,0,5"TextWrap
ping="Wrap"Text="TextBox"VerticalAlignment="Bottom"Width="678"FontSize="20"Grid.Row="1"B
orderBrush="{x:Null}"/>
<GridHeight="600"Margin="296,53,314,74"Grid.Row="1"Width="600">

<EllipseName
="LeftIris"Fill="#FFEAEAFB"HorizontalAlignment="Left"Height="40"Margin="180,254,0,0"Stroke="
Black"VerticalAlignment="Top"Width="40"/>
<EllipseName
="LeftPupil"Fill="Black"HorizontalAlignment="Left"Height="20"Margin="190,264,0,0"Stroke="Black
"VerticalAlignment="Top"Width="20"/>
<EllipseName="RightIris"Fill="#FFEAEAFB"HorizontalAlignment="Left"Height="40"Margin="380,2
54,0,0"Stroke="Black"VerticalAlignment="Top"Width="40"/>
<EllipseName="RightPupil"Fill="Black"HorizontalAlignment="Left"Height="20"Margin="390,264,0,0
"Stroke="Black"VerticalAlignment="Top"Width="20"/>

<GridName="faceGrid"Height="600"Width="600"Margin="0,12,0,-12">
<Grid.Background>
<ImageBrushImageSource="Images/Sleeping.png"/>
</Grid.Background>
</Grid>

</Grid>
<LabelName="GestureRecognized"Visibility="Hidden"Content="!"HorizontalAlignment="Left"Height
="281"Margin="1057,116,0,0"Grid.Row="1"VerticalAlignment="Top"Width="170"FontSize="200"Ba
ckground="{x:Null}"Foreground="#FF783232"RenderTransformOrigin="0.5,0.5">
<Label.RenderTransform>
<TransformGroup>
<ScaleTransform/>
<SkewTransform/>
<RotateTransformAngle="11.31"/>
<TranslateTransform/>
</TransformGroup>
</Label.RenderTransform>
</Label>

</Grid>
</Window>
```

19.6.2 MainWindows.xaml.cs

```
namespace KinectAGVNavigator
{
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Diagnostics;
using System.Globalization;
using System.IO;
using System.Windows;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using Microsoft.Kinect;
using System.Linq;
using System.Windows.Controls;
using System.Collections.Specialized;
using System.IO.Ports;


/// <summary>
/// Interaction logic for MainWindow
/// </summary>
public partial class MainWindow : Window, INotifyPropertyChanged
  {
/// <summary>
/// Radius of drawn hand circles
/// </summary>
privateconstdouble HandSize = 30;

/// <summary>
/// Thickness of drawn joint lines
/// </summary>
privateconstdouble JointThickness = 3;


/// <summary>
/// Thickness of clip edge rectangles
/// </summary>
privateconstdouble ClipBoundsThickness = 10;

/// <summary>
/// Constant for clamping Z values of camera space points from being negative
/// </summary>
privateconstfloat InferredZPositionClamp = 0.1f;

/// <summary>
/// Brush used for drawing hands that are currently tracked as closed
/// </summary>
privatereadonly Brush handClosedBrush = new SolidColorBrush(Color.FromArgb(128, 255, 0, 0));

/// <summary>
```

```
/// Brush used for drawing hands that are currently tracked as opened
/// </summary>
privatereadonly Brush handOpenBrush = new SolidColorBrush(Color.FromArgb(128, 0, 255, 0));

/// <summary>
/// Brush used for drawing hands that are currently tracked as in lasso (pointer) position
/// </summary>
privatereadonly Brush handLassoBrush = new SolidColorBrush(Color.FromArgb(128, 0, 0, 255));

/// <summary>
/// Brush used for drawing joints that are currently tracked
/// </summary>
privatereadonly Brush trackedJointBrush = new SolidColorBrush(Color.FromArgb(255, 68, 192, 68));

/// <summary>
/// Brush used for drawing joints that are currently inferred
/// </summary>
privatereadonly Brush inferredJointBrush = Brushes.Yellow;

/// <summary>
/// Pen used for drawing bones that are currently inferred
/// </summary>
privatereadonly Pen inferredBonePen = new Pen(Brushes.Gray, 1);

/// <summary>
/// Drawing group for body rendering output
/// </summary>
private DrawingGroup drawingGroup;

/// <summary>
/// Drawing image that we will display
/// </summary>
private DrawingImage imageSource;

/// <summary>
/// Active Kinect sensor
/// </summary>
private KinectSensor kinectSensor = null;

/// <summary>
/// Coordinate mapper to map one type of point to another
/// </summary>
private CoordinateMapper coordinateMapper = null;

/// <summary>
/// Reader for body frames
/// </summary>
private BodyFrameReader bodyFrameReader = null;

/// <summary>
/// MultriFrame Reader
/// </summary>
private MultiSourceFrameReader multiSourceFrameReader = null;
```

```
/// <summary>
/// Array for the bodies
/// </summary>
private Body[] bodies = null;

/// <summary>
/// definition of bones
/// </summary>
private List<Tuple<JointType, JointType>> bones;

/// <summary>
/// Width of display (depth space)
/// </summary>
privateint displayWidth;

/// <summary>
/// Height of display (depth space)
/// </summary>
privateint displayHeight;

/// <summary>
/// List of colors for each body tracked
/// </summary>
private List<Pen> bodyColors;

/// <summary>
/// Current status text to display
/// </summary>
privatestring statusText = null;

/// <summary>
/// User Information
/// </summary>
private UserData user = new UserData();

/// <summary>
/// List of possible players
/// </summary>
private List<String> Players = new List<String> { "Player 1", "Player 2", "Player 3", "Player 4", "Player
5", "Player 6" };

/// <summary>
/// Dynamic Time Warp object
/// </summary>
private DynamicTimeWarp DTW = new DynamicTimeWarp();

/// <summary>
/// Memory space for color image
/// </summary>
privatebyte[] pixels = newbyte[1920 * 1080 * 4];

/// <summary>
```

```
/// Timer for color refershing
/// </summary>
    Stopwatch colorRefresh = new Stopwatch();

/// <summary>
/// Gesture manager object
/// </summary>
private GestureManager gestureManager = new GestureManager();

// BORRAR
//private bool ListViewRefreshed = false;

/// <summary>
/// List of progressbars to show recognition level for each gesture
/// </summary>
private List<ProgressBar> ProgressBarList = new List<ProgressBar>();


/// <summary>
/// List of player datas
/// </summary>
private List<PlayerData> PlayerDataList = new List<PlayerData>() { new PlayerData(0), new
PlayerData(1), new PlayerData(2), new PlayerData(3), new PlayerData(4), new PlayerData(5)};

/// <summary>
/// Serial Port
/// </summary>
    SerialPort serialPort = new SerialPort();

/// <summary>
/// Timer for Serial Port
/// </summary>
    System.Windows.Threading.DispatcherTimer comPortSendTimer = new
System.Windows.Threading.DispatcherTimer();

/// <summary>
/// Navigation state object
/// </summary>
    NavigatorState navState = new NavigatorState();

/// <summary>
/// AGV state object
/// </summary>
    AgvState agvState = new AgvState();

/// <summary>
/// Virtual assistant state object
/// </summary>
    VirtualAssistantState virtualAssistantState = new VirtualAssistantState();

/// <summary>
/// Memory space to store message
/// </summary>
```

```
string message;

/// <summary>
/// Initializes a new instance of the MainWindow class.
/// </summary>
public MainWindow()
    {


// one sensor is currently supported
this.kinectSensor = KinectSensor.GetDefault();

// get the coordinate mapper
this.coordinateMapper = this.kinectSensor.CoordinateMapper;

// get the depth (display) extents
        FrameDescription frameDescription = this.kinectSensor.DepthFrameSource.FrameDescription;

// get size of joint space
this.displayWidth = frameDescription.Width;
this.displayHeight = frameDescription.Height;

// open the reader for the body frames
this.bodyFrameReader = this.kinectSensor.BodyFrameSource.OpenReader();

// open the reader for the multi source frames
this.multiSourceFrameReader                                                                    =
this.kinectSensor.OpenMultiSourceFrameReader(FrameSourceTypes.Color                             |
FrameSourceTypes.Body);


// a bone defined as a line between two joints
this.bones = new List<Tuple<JointType, JointType>>();

// Torso
this.bones.Add(new Tuple<JointType, JointType>(JointType.Head, JointType.Neck));
this.bones.Add(new Tuple<JointType, JointType>(JointType.Neck, JointType.SpineShoulder));
this.bones.Add(new Tuple<JointType, JointType>(JointType.SpineShoulder, JointType.SpineMid));
this.bones.Add(new Tuple<JointType, JointType>(JointType.SpineMid, JointType.SpineBase));
this.bones.Add(new                Tuple<JointType,                JointType>(JointType.SpineShoulder,
JointType.ShoulderRight));
this.bones.Add(new Tuple<JointType, JointType>(JointType.SpineShoulder, JointType.ShoulderLeft));
this.bones.Add(new Tuple<JointType, JointType>(JointType.SpineBase, JointType.HipRight));
this.bones.Add(new Tuple<JointType, JointType>(JointType.SpineBase, JointType.HipLeft));

// Right Arm
this.bones.Add(new Tuple<JointType, JointType>(JointType.ShoulderRight, JointType.ElbowRight));
this.bones.Add(new Tuple<JointType, JointType>(JointType.ElbowRight, JointType.WristRight));
this.bones.Add(new Tuple<JointType, JointType>(JointType.WristRight, JointType.HandRight));
this.bones.Add(new Tuple<JointType, JointType>(JointType.HandRight, JointType.HandTipRight));
this.bones.Add(new Tuple<JointType, JointType>(JointType.WristRight, JointType.ThumbRight));

// Left Arm
```

```
this.bones.Add(new Tuple<JointType, JointType>(JointType.ShoulderLeft, JointType.ElbowLeft));
this.bones.Add(new Tuple<JointType, JointType>(JointType.ElbowLeft, JointType.WristLeft));
this.bones.Add(new Tuple<JointType, JointType>(JointType.WristLeft, JointType.HandLeft));
this.bones.Add(new Tuple<JointType, JointType>(JointType.HandLeft, JointType.HandTipLeft));
this.bones.Add(new Tuple<JointType, JointType>(JointType.WristLeft, JointType.ThumbLeft));

// Right Leg
this.bones.Add(new Tuple<JointType, JointType>(JointType.HipRight, JointType.KneeRight));
this.bones.Add(new Tuple<JointType, JointType>(JointType.KneeRight, JointType.AnkleRight));
this.bones.Add(new Tuple<JointType, JointType>(JointType.AnkleRight, JointType.FootRight));

// Left Leg
this.bones.Add(new Tuple<JointType, JointType>(JointType.HipLeft, JointType.KneeLeft));
this.bones.Add(new Tuple<JointType, JointType>(JointType.KneeLeft, JointType.AnkleLeft));
this.bones.Add(new Tuple<JointType, JointType>(JointType.AnkleLeft, JointType.FootLeft));

// populate body colors, one for each BodyIndex
this.bodyColors = new List<Pen>();

this.bodyColors.Add(new Pen(Brushes.Red, 6));
this.bodyColors.Add(new Pen(Brushes.Orange, 6));
this.bodyColors.Add(new Pen(Brushes.Green, 6));
this.bodyColors.Add(new Pen(Brushes.Blue, 6));
this.bodyColors.Add(new Pen(Brushes.Indigo, 6));
this.bodyColors.Add(new Pen(Brushes.Violet, 6));

// set IsAvailableChanged event notifier
this.kinectSensor.IsAvailableChanged += this.Sensor_IsAvailableChanged;

// open the sensor
this.kinectSensor.Open();

// set the status text
this.StatusText = this.kinectSensor.IsAvailable ? Properties.Resources.RunningStatusText
                                                : Properties.Resources.NoSensorStatusText;

// Create the drawing group we'll use for drawing
this.drawingGroup = new DrawingGroup();

// use the window object as the view model in this simple example
this.DataContext = this;

// initialize the components (controls) of the window
this.InitializeComponent();

    }

/// <summary>
/// INotifyPropertyChangedPropertyChanged event to allow window controls to bind to changeable
data
/// </summary>
publicevent PropertyChangedEventHandler PropertyChanged;
```

```
/// <summary>
/// Gets or sets the current status text to display
/// </summary>
publicstring StatusText
    {
      get
      {
returnthis.statusText;
      }

      set
      {
if (this.statusText != value)
        {
this.statusText = value;

// notify any bound elements that the text has changed
if (this.PropertyChanged != null)
          {
this.PropertyChanged(this, new PropertyChangedEventArgs("StatusText"));
          }
        }
      }
    }

/// <summary>
/// Execute start up tasks
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
privatevoid MainWindow_Loaded(object sender, RoutedEventArgs e)
    {
// Loads Database into RAM
      DataBaseInit();

// Starts the timer for color refreshing
      colorRefresh.Start();


if (this.multiSourceFrameReader != null)
      {

this.multiSourceFrameReader.MultiSourceFrameArrived += this.Reader_MultiSourceFrameArrived;
      }

// gets list of available serial ports
      var serialPortList = SerialPort.GetPortNames();

// if a serial port was found take the first one, configure it to 9600bauds and open it
if (serialPortList.Count() >0)
      {
        serialPort.PortName = serialPortList[0];
```

```
            serialPort.BaudRate = 9600;
            serialPort.Open();
        }
// Event handler for data recived event in serial port
        serialPort.DataReceived += this.comPortDataRecieved;

// Configure timer for data sending in serial port. Send data every 100ms
        comPortSendTimer.Tick += new EventHandler(comPortSendTimer_Tick);
        comPortSendTimer.Interval = new TimeSpan(0, 0, 0, 0, 100);
        comPortSendTimer.Start();

    }

/// <summary>
/// Execute shutdown tasks
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
privatevoid MainWindow_Closing(object sender, CancelEventArgs e)
    {
if (this.multiSourceFrameReader != null)
        {
// Close frame reader
this.multiSourceFrameReader.Dispose();
this.multiSourceFrameReader = null;
        }

if (this.kinectSensor != null)
        {
// Close kinect sensor
this.kinectSensor.Close();
this.kinectSensor = null;
        }

if (serialPort.IsOpen)
        {
// Close serial port
            serialPort.Close();
        }
    }

/// <summary>
/// Initiates Database
/// </summary>
publicvoid DataBaseInit()
    {
// Gets database path
        var dbpath = System.IO.Path.Combine(Directory.GetCurrentDirectory(),
"GestureDatabase.db3");
using (var db = new SQLite.SQLiteConnection(dbpath))
        {
// Create the tables if they don't exist
            db.CreateTable<DBgesture>();
```

```
            db.Commit();
// Loops around the database
foreach (var sd in db.Table<DBgesture>())
            {
string name = sd.name;
//double[] ShY = sd.ShY.Split(',').Select(double.Parse).ToArray();
double[] ShY = Array.ConvertAll(sd.ShY.Split(','), double.Parse);
double[] ShP = sd.ShP.Split(',').Select(double.Parse).ToArray();
double[] ShR = sd.ShR.Split(',').Select(double.Parse).ToArray();
double[] ElP = sd.ElP.Split(',').Select(double.Parse).ToArray();
double[] ElR = sd.ElR.Split(',').Select(double.Parse).ToArray();
double[] HaP = sd.HaP.Split(',').Select(double.Parse).ToArray();

// if Gesture is first of its kind create new Gesture in the gesture list
if (!gestureManager.GestureAllreadyExists(name))
                {
                   gestureManager.newGestureType(name);
                }

// store gesture data
for (int i = 0; i <30; i++)
                {
double[] ang = { ShY[i], ShP[i], ShR[i], ElP[i], ElR[i], HaP[i] };
                   gestureManager.newSequence(name, ang);
                }
// end gesture data
                gestureManager.stopNewSequence(name);
            }

            db.Dispose();
            db.Close();

        }
// create a Gesture likelyhood visualizer for each Gesture (DEBUG ONLY)
foreach (Gesture Gestures in gestureManager.Gestures)
        {
// add a text with the title of the gesture
            GestureRecognitionNames.Items.Add(new TextBlock() { Text = Gestures.GestureName,
Foreground = new SolidColorBrush(Colors.Black), Width = 200 });

// add a progressbar for the gesture
            ProgressBarList.Add(new ProgressBar() { Name = Gestures.GestureName, Value = 0, Width =
200 });

// if the gesture's name starts with "o" (order)
if (Gestures.GestureName.StartsWith("o")){

// add an entry in the navigation state's detection dictionary
                navState.detection.Add(Gestures.GestureName, false);
            }

        }
// source the GestureReconitionLevel item
```

```
        GestureRecognitionLevel.ItemsSource = ProgressBarList;

    }


/// <summary>
/// Handles the body frame data arriving from the sensor
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
privatevoid Reader_MultiSourceFrameArrived(object sender, MultiSourceFrameArrivedEventArgs e)
    {
// data received flag
bool dataReceived = false;

// refresh user data flag
bool refreshUsers = false;

// tracked body state
bool[] bodyTrackingStates = newbool[6] { false, false, false, false, false, false };

// player engaging
bool[] playerStarting = newbool[6] { false, false, false, false, false, false };

// get frame
        var reference = e.FrameReference.AcquireFrame();

// each second
if (colorRefresh.ElapsedMilliseconds >1000)
        {
//get color frame
using (ColorFrame colorFrame = reference.ColorFrameReference.AcquireFrame())
          {
// if color frame is available
if (colorFrame != null)
            {
// extract frame description
                FrameDescription colorFrameDescription = colorFrame.FrameDescription;

// verify data and write the new color frame data to the Writeable bitmap
if ((colorFrameDescription.Width == 1920) && (colorFrameDescription.Height == 1080))
                {
if (colorFrame.RawColorImageFormat == ColorImageFormat.Bgra)
                  {
                    colorFrame.CopyRawFrameDataToArray(pixels);
                  }
else
                  {
                    colorFrame.CopyConvertedFrameDataToArray(pixels, ColorImageFormat.Bgra);
                  }
                }
// update userColor
```

```
int[] userColor = getUserColor();

//if color is correct
if (userColor[0] <256)
                {
                   user.color = userColor;
                }
// activate refresh users flag
                refreshUsers = true;

// resets the timer
                colorRefresh.Restart();
            }
          }
        }

// get body frame
using (BodyFrame bodyFrame = reference.BodyFrameReference.AcquireFrame())
        {
// if body frame is available
if (bodyFrame != null)
            {
// defines space to alocate body information
if (this.bodies == null)
                {
this.bodies = new Body[bodyFrame.BodyCount];
                }

// The first time GetAndRefreshBodyData is called, Kinect will allocate each Body in the array.
// As long as those body objects are not disposed and not set to null in the array,
// those body objects will be re-used.
                bodyFrame.GetAndRefreshBodyData(this.bodies);

// activate data received flag
                dataReceived = true;
            }
        }

// if data was received
if (dataReceived)
        {

// body index
int bodyIndex = 0;

// loop through all bodies
foreach (Body body inthis.bodies)
            {
// if the body is being tracked
if (body.IsTracked)
                {
// if a color frame was taken in this iteration (refreshUser == true) and a former user is being searched
for
```

```
if (refreshUsers && user.searchForUser)
                {
// update color data for body
                    user.updatePosibleUserColor(bodyIndex, getPlayerColor(body));
                }
// change tracking state of body
                bodyTrackingStates[bodyIndex] = true;

// update player data
                PlayerDataList[bodyIndex].update(body);

// if the analyzed player is not the user and the player has performed the "Start" gesture
if (!(user.isTracked && user.PlayerID == bodyIndex) &&
DTW.identifySpecificGesturePlayer(PlayerDataList[bodyIndex], gestureManager, "s_Start"))
                {
// change player starting status
                    playerStarting[bodyIndex] = true;
                }

            }
else//body is not being tracked
            {
// if a color frame was taken in this iteration (refreshUser == true)
if (refreshUsers)
                {
// update body color with "no color"
int[] noColor = newint[] { 300, 300, 300 };
                    user.updatePosibleUserColor(bodyIndex, noColor);
                }
            }
// increase body index
            bodyIndex++;
          }
        }

// ##################
// #  USER UPDATE  #
// ##################

// update user selection
        user.selectUser(bodyTrackingStates, playerStarting, navState);

// Update virtual magnetic sensor
        user.updateMacome(navState);

// update face and eyes
        updateEyes(virtualAssistantState.update(agvState, navState, user, bodyTrackingStates,
faceGrid));


// if user is being tracked
if (user.isTracked)
        {
```

```
// update joint information
            user.update(bodies[user.PlayerID]);

// get velocity in x and y axis
            user.computeVelocity();

// update the speed reference for the controller
            navState.speedSetPointUpdate(agvState,user);

//  identify gestures
            DTW.identifyGesture(user, gestureManager, ProgressBarList, navState);

// show "gesture recognized" exclamation if a gesture was recognized, else hide it
if              (DTW.ignoreGestureRecognition)              {              GestureRecognized.Visibility              =
System.Windows.Visibility.Visible; }
else {GestureRecognized.Visibility = System.Windows.Visibility.Hidden;}

        }
    }

/// <summary>
/// Gets user's t-Shirt color.
/// </summary>
int[] getUserColor()
    {
// declare color variables
int red = 256;
int green = 256;
int blue = 256;

// if user is being tracked
if (user.isTracked)
        {
// resets color variables
            red = 0;
            green = 0;
            blue = 0;

// get 2D-image coordinates of body
            ColorSpacePoint                                    UserPosition                                    =
kinectSensor.CoordinateMapper.MapCameraPointToColorSpace(user.BodySpacePoint);

// if a valid distance is obtained
if (user.BodySpacePoint.Z >0)
            {
// define square size
int squareSide = Convert.ToInt16(160 / user.BodySpacePoint.Z);

// if square size is bigger than 500x500 pixels, reduce it to 500x500
if (squareSide >500) squareSide = 500;

// if square fits into image
```

```
if (UserPosition.X > squareSide && UserPosition.X < (1920 - (squareSide)) && UserPosition.Y > squareSide && UserPosition.Y <1080 - (squareSide))
            {
// convert coordinates into integers
int xCoor = Convert.ToInt16(UserPosition.X);
int yCoor = Convert.ToInt16(UserPosition.Y);

// loop through each pixel of square
for (int x = xCoor - squareSide / 2; x < xCoor + squareSide / 2; x++)
                {
for (int y = yCoor - squareSide / 2; y < yCoor + squareSide / 2; y++)
                  {
// sum all pixel values
                    red = red + pixels[4 * x + (1920 * 4) * y + 2];
                    green = green + pixels[4 * x + (1920 * 4) * y + 1];
                    blue = blue + pixels[4 * x + (1920 * 4) * y + 0];
                  }

                }

// divide sum of values by its number of elements (mean)
                red = red / (squareSide * squareSide);
                green = green / (squareSide * squareSide);
                blue = blue / (squareSide * squareSide);

// refresh color of colorbox
                ColorBox.Fill = new SolidColorBrush(Color.FromArgb(255, Convert.ToByte(red), Convert.ToByte(green), Convert.ToByte(blue)));
            }
          }
        }

// copy color into array
int[] color = newint[3] { red, green, blue };

// return color
return color;
      }

/// <summary>
/// Gets players' t-shirt color.
/// </summary>
/// <param name="player" Body information of player to scan </param>
int[] getPlayerColor(Body player)
      {
// declare color variables
int red = 0;
int green = 0;
int blue = 0;

// if player is being tracked
if (player.IsTracked)
        {
```

```
// if distance to player's spine base is valid
if (player.Joints[JointType.SpineBase].Position.Z >0)
        {
// define square size
int squareSide = Convert.ToInt16(160 / player.Joints[JointType.SpineBase].Position.Z);

// if square size is bigger than 500x500 pixels, reduce it to 500x500
if (squareSide >500) squareSide = 500;

// get 2D-image coordinates of spinde base
            ColorSpacePoint                                    UserPosition                                    =
kinectSensor.CoordinateMapper.MapCameraPointToColorSpace(player.Joints[JointType.SpineBase].Po
sition);

// if square fits into image
if (UserPosition.X > squareSide && UserPosition.X < (1920 - (squareSide)) && UserPosition.Y >
squareSide && UserPosition.Y <1080 - (squareSide))
            {
// convert coordinates into integers
int xCoor = Convert.ToInt16(UserPosition.X);
int yCoor = Convert.ToInt16(UserPosition.Y);

// loop through each pixel of square
for (int x = xCoor - squareSide / 2; x < xCoor + squareSide / 2; x++)
              {
for (int y = yCoor - squareSide / 2; y < yCoor + squareSide / 2; y++)
{
// sum all pixel values
                    red = red + pixels[4 * x + (1920 * 4) * y + 2];
                    green = green + pixels[4 * x + (1920 * 4) * y + 1];
                    blue = blue + pixels[4 * x + (1920 * 4) * y + 0];
                }

              }

// divide sum of values by its number of elements (mean)
              red = red / (squareSide * squareSide);
              green = green / (squareSide * squareSide);
              blue = blue / (squareSide * squareSide);
            }
        }
      }

// copy color into array
int[] color = newint[3] { red, green, blue };

// return color
return color;
    }

/// <summary>
/// Updates eyes position
/// </summary>
```

```
/// <param name="position" User position </param>
privatevoid updateEyes(int[] position){

// update left iris position
        LeftIris.Margin = new Thickness() { Left = position[0], Top = position[1]};

// update left pupil position
        LeftPupil.Margin = new Thickness() { Left = position[0]+10, Top = position[1]+10 };

// update right iris position
        RightIris.Margin = new Thickness() { Left = position[2], Top = position[3] };

// update right pupil position
        RightPupil.Margin = new Thickness() { Left = position[2] + 10, Top = position[3] + 10 };
}


/// <summary>
/// Handles inputs of serial port
/// </summary>
privatevoid comPortDataRecieved(object sender, EventArgs e)
{
try
        {
//AWARDVVVBBB
//||||| |  └>Baterry
//|||||  └>Velocity
//||||  └>Direction
//|||  └>Run
//||  └>Armed
//|  └>Warning
//  └>Alarm

// read line of serial port
            message = serialPort.ReadLine();

// show message on screen
            SerialPortText.Text = message;

// interpret message
if (message[0] == '0') agvState.alarm = false;
elseif (message[0] == '1') agvState.alarm = true;

if (message[1] == '0') agvState.warning = false;
elseif (message[1] == '1') agvState.warning = true;

if (message[2] == '0') agvState.armed = false;
elseif (message[2] == '1') agvState.armed = true;

if (message[3] == '0') agvState.run = false;
elseif (message[3] == '1') agvState.run = true;

if (message[4] == '0') agvState.backwards = false;
```

```
elseif (message[4] == '1') agvState.backwards = true;

            agvState.speed = Convert.ToInt16(message[5].ToString()) * 100 +
Convert.ToInt16(message[6].ToString()) * 10 + Convert.ToInt16(message[7].ToString());
            agvState.baterry = Convert.ToInt16(message[8].ToString()) * 100 +
Convert.ToInt16(message[9].ToString()) * 10 + Convert.ToInt16(message[10].ToString());
        }
catch (TimeoutException) { }
    }

/// <summary>
/// Send periodically message
/// </summary>
privatevoid comPortSendTimer_Tick(object sender, EventArgs e)
    {
// if serial port is open
if (serialPort.IsOpen)
        {
// send status
            serialPort.Write(navState.getStatus());
        }
        SerialPortText.Text = navState.getStatus();
    }


/// <summary>
/// Handles the event which the sensor becomes unavailable (E.g. paused, closed, unplugged).
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
privatevoid Sensor_IsAvailableChanged(object sender, IsAvailableChangedEventArgs e)
    {
// on failure, set the status text
this.StatusText = this.kinectSensor.IsAvailable ? Properties.Resources.RunningStatusText
                                    : Properties.Resources.SensorNotAvailableStatusText;
}
  }
}
```

19.6.3 AgvState.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace KinectAGVNavigator
{
class AgvState
   {
// AGV's speed
publicint speed = 0;

// AGV's battery level
publicint baterry = 0;

// AGV's alarm status
publicbool alarm = false;

// AGV's warning status
publicbool warning = false;

// AGV's armed status
publicbool armed = false;

// AGV's running status
publicbool run = false;

// AGV's moving direction
publicbool backwards = false;

}
}
```

19.6.4 DBGesture.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SQLite; // for sqlite

namespace KinectAGVNavigator
{
publicclass DBgesture
   {
// gesture title
      [MaxLength(20)]
public String name { get; set; }

// shoulder yaw
      [MaxLength(511)]
public String ShY { get; set; }

// shoulder pitch
      [MaxLength(511)]
public String ShP { get; set; }

// shoulder roll
      [MaxLength(511)]
public String ShR { get; set; }

// elbow pitch
      [MaxLength(511)]
public String ElP { get; set; }

// elbow roll
      [MaxLength(511)]
public String ElR { get; set; }

// hand pitch
      [MaxLength(511)]
public String HaP { get; set; }

}
}
```

19.6.5 DynamicTimeWarp.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Controls;
using System.Windows.Media;
using System.Diagnostics;

namespace KinectAGVNavigator
{
class DynamicTimeWarp
  {

// Timer to block gesture recognition process after a gesture has been recognized
    Stopwatch ignoreGestureRecognitionTime = new Stopwatch();

// Flag to block gesture recognition process after a gesture has been recognized
publicbool ignoreGestureRecognition = false;

// Name of the engange gesture
string engangeGesture = "s_Start";

/// <summary>
/// Identifies gestures of the user
/// </summary>
/// <param name="user"> User data </param>
/// <param name="gestureManager"> Gesture manager </param>
/// <param name="PBL"> Progress bar list </param>
/// <param name="navState"> Navigation state </param>
publicvoid identifyGesture(UserData user, GestureManager gestureManager, List<ProgressBar> PBL,
NavigatorState navState)
    {
// if block gesture recognition process is active
if (ignoreGestureRecognition)
      {
// if 2 seconds have passed since the last recognized gesture
if (ignoreGestureRecognitionTime.ElapsedMilliseconds >2000)
        {
// reset timer
          ignoreGestureRecognitionTime.Reset();

// reset flag to block gesture recognition process
          ignoreGestureRecognition = false;
        }
      }
else// if gesture recognition process is not blocked
      {
// foreach gesture
foreach (Gesture gestureType in gestureManager.Gestures)
```

```
            {
// if the user is facing the camera
if (user.isUserFacingToCamera())
                {
// define and reset sequence index
int index = 0;

// create an array to store the results for each sequence
double[] DTWvalues = newdouble[] { 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100 };

// for each sequence in gesture
foreach (GestureSequence sequence in gestureType.GestureSequences)
                {
// compare user gesture to sequence
double result = compareGesture(user, sequence);

// if the gestures may be the same (if they are to distant compareGesture returns 1000000000)
if ( result <1000000000)
                    {
// parese result on index position of results array
                        DTWvalues[index] = result;

// increase index
                        index = index + 1;
                    }

                }

// search for the progressbar related to the gesture
                ProgressBar PB = PBL.Find(x=>x.Name == gestureType.GestureName);

// if at least one sequence of the gesture was similar to the user's gesture
if (index >0)
                {
// find the minimal result
double minDTW = DTWvalues.Min();

// scale the result such that 0 -> 100% and 30 -> 0%
                    var score = (30 - minDTW) * 3.33;

// if the score is higher than 0 copy it on the progressbar
if (score >0) PB.Value = score;

// else copy 0
else PB.Value = 0;

// if the score was over 70% (the DTW distance is lower than ~9)
if (score >70)
```

```
                {

// change progressbar's color to red
                    PB.Foreground = new SolidColorBrush(Colors.Red);

// activate block gesture recognition process flag
                    ignoreGestureRecognition = true;

// start block gesture recognition process timer
                    ignoreGestureRecognitionTime.Start();

// if the gesture title started with "o" (order gesture)
if (gestureType.GestureName.StartsWith("o"))
                    {
// search for it in the navigation detection status and set it to true
                      navState.detection[gestureType.GestureName] = true;
                    }

// if the performed gesture is the engange gesture
if (gestureType.GestureName == engangeGesture)
                    {
// stop user tracking
                      user.isTracked = false;

// do not search for the user
                      user.searchForUser = false;

// send a stop order
                      navState.detection["o_Stop"] = true;
                    }
                  }
// else, if the score is under 70% change, change progressbar's color to violet
else PB.Foreground = new SolidColorBrush(Color.FromArgb(255, 91, 46, 197));

                }
else// else, if no similar sequence was detected
                {
// show on the progressbar a score of 0
                  PB.Value = 0;

// change progressbar's color to violet
                  PB.Foreground = new SolidColorBrush(Color.FromArgb(255, 91, 46, 197));
                }
              }
else// else, if the user is not facing to the camera
              {
// find the progressbar of the analyzed gesture
                ProgressBar PB = PBL.Find(x => x.Name == gestureType.GestureName);

// show on the progressbar a score of 0
                PB.Value = 0;

// change progressbar's color to violet
```

```
                PB.Foreground = new SolidColorBrush(Color.FromArgb(255, 91, 46, 197));
            }
        }
    }
}


/// <summary>
/// Identifies a specific gesture of the user.
/// This functions is not being used.
/// </summary>
/// <param name="user"> User data </param>
/// <param name="gestureManager"> Gesture manager </param>
/// <param name="gestureName"> The name of the gesture to be analyzed </param>
publicbool identifySpecificGestureUser(UserData user, GestureManager gestureManager, string
gestureName)
    {
bool gestureIdentified = false;
        Gesture gesture = gestureManager.Gestures.Find(x => x.GestureName == gestureName);

if (ignoreGestureRecognition)
        {
if (ignoreGestureRecognitionTime.ElapsedMilliseconds >2000)
            {
                ignoreGestureRecognitionTime.Reset();
                ignoreGestureRecognition = false;
            }
        }
else
        {
if (user.isUserFacingToCamera())
            {
int index = 0;
double[] DTWvalues = newdouble[] { 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100 };

foreach (GestureSequence sequence in gesture.GestureSequences)
                {
if (compareGesture(user, sequence) <1000000000)
                    {
                        DTWvalues[index] = compareGesture(user, sequence);
                        index = index + 1;
                    }
                }

if (index >0)
                {
double minDTW = DTWvalues.Min();
                    var value = (30 - minDTW) * 3.33;
```

```
if (value >70)
                {
                   gestureIdentified = true;
                   ignoreGestureRecognition = true;
                   ignoreGestureRecognitionTime.Start();
                }
             }
          }
       }
return gestureIdentified;

    }


/// <summary>
/// Identifies a specific gesture of the player
/// </summary>
/// <param name="player"> Player to be analyzed </param>
/// <param name="gestureManager"> Gesture manager </param>
/// <param name="gestureName"> Gesture's name </param>
publicbool identifySpecificGesturePlayer(PlayerData player, GestureManager gestureManager, string
gestureName)
    {
// set gesture identified flag to false
bool gestureIdentified = false;

// find the desired gesture
       Gesture gesture = gestureManager.Gestures.Find(x => x.GestureName == gestureName);

// if block gesture recognition flag is active
if (ignoreGestureRecognition)
       {
// if 2 seconds have pases since the last recognition
if (ignoreGestureRecognitionTime.ElapsedMilliseconds >2000)
          {
// reset timer
             ignoreGestureRecognitionTime.Reset();

// reset flag to block gesture recognition process
             ignoreGestureRecognition = false;
          }
       }
else// if gesture recognition process is not blocked
       {
// if player is facing to camera
if (player.isUserFacingToCamera())
          {
// define and reset sequence index
int index = 0;

// create an array to store the results for each sequence
double[] DTWvalues = newdouble[] { 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
```

```
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100 };

// for each sequence in gesture
foreach (GestureSequence sequence in gesture.GestureSequences)
            {
// compare player's gesture to sequence
double result = compareGesture(player, sequence);

// if the gestures may be the same (if they are to distant compareGesture returns 1000000000)
if (result <1000000000)
              {
// parese result on index position of results array
                DTWvalues[index] = result;

// increase index
                index = index + 1;
              }

            }

// if at least one sequence of the gesture was similar to the user's gesture
if (index >0)
            {

// find the minimal result
double minDTW = DTWvalues.Min();

// scale the result such that 0 -> 100% and 30 -> 0%
              var score = (30 - minDTW) * 3.33;

// if the score is higher than 70%
if (score >70)
              {
// activate gesture identified flag
                gestureIdentified = true;

// activate block gesture recognition process flag
                ignoreGestureRecognition = true;

// start block gesture recognition process timer
                ignoreGestureRecognitionTime.Start();
              }

            }

          }
        }

// return gesture identified flag
return gestureIdentified;
```

```
    }

/// <summary>
/// Compares the user's movement to a gestures
/// </summary>
/// <param name="user"> User data </param>
/// <param name="gesture"> Gesture to be compared to </param>
double compareGesture(UserData user, GestureSequence gesture)
    {
// flag to indicate that both gesture may be similar
bool couldBeSame = true;

// for each feature of the sequence
for (int i = 0; i <4; i++)
      {
// if means have a diference higher than 0.25
if (Math.Abs(user.Sequence.means[i] - gesture.means[i]) >0.25)
        {
// put flag to indicate that both gesture may be similar to false
          couldBeSame = false;

// break the loop
break;
        }
      }
// if gesture may be the same
if (couldBeSame)
      {
// return their DTW distance
return DTW(user.ACSequence.Angles, gesture.Angles);
      }
// else return 1000000000
elsereturn1000000000;
    }


/// <summary>
/// Compares the player's movement to a gesture
/// </summary>
/// <param name="player"> User data </param>
/// <param name="gesture"> Gesture to be compared to </param>
double compareGesture(PlayerData player, GestureSequence gesture)
    {
// flag to indicate that both gesture may be similar
bool couldBeSame = true;

// for each feature of the sequence
for (int i = 0; i <4; i++)
      {
// if means have a diference higher than 0.25
if (Math.Abs(player.Sequence.means[i] - gesture.means[i]) >0.25)
```

```
            {
// put flag to indicate that both gesture may be similar to false
                couldBeSame = false;

// break the loop
break;
            }
        }
// if gesture may be the same
if (couldBeSame)
        {
// return their DTW distance
return DTW(player.ACSequence.Angles, gesture.Angles);
        }
// else return 1000000000
elsereturn1000000000;
    }

/// <summary>
/// Performs DTW algorithm between gesture A and gesture B
/// </summary>
/// <param name="gestureA"> Gesture A data </param>
/// <param name="gestureB"> Gesture B data </param>
double DTW(double[,] gestureA, double[,] gestureB)
    {

//Window size
int w = 10;

//DTW matrix initizalization
double[,] DTWMatrix = newdouble[30, 30];

// for each cell in the matrix
for (int i = 0; i <30; i++)
        {
for (int j = 0; j <30; j++)
            {
// put cell's value to "infinite" (a very high number)
                DTWMatrix[i, j] = 1000000;
            }

        }

// First value is just the cost of diferences
// The cost is the square root of the sum of the squares of the diference between each feature of gesture
A and gesture B
        DTWMatrix[0, 0] = Math.Sqrt((gestureA[0, 0] - gestureB[0, 0]) * (gestureA[0, 0] - gestureB[0, 0])
+
                        (gestureA[0, 1] - gestureB[0, 1]) * (gestureA[0, 1] - gestureB[0, 1]) +
                        (gestureA[0, 2] - gestureB[0, 2]) * (gestureA[0, 2] - gestureB[0, 2]) +
                        (gestureA[0, 3] - gestureB[0, 3]) * (gestureA[0, 3] - gestureB[0, 3]));

// for each x coordinate
```

```
for (int i = 1; i <30; i++)
        {
// define starting y coordinate respect to x and window size
int start = (i - w <1) ? (1) : (i - w);
// define ending y coordinate respect to x and window size
int end = (i + w + 1>30) ? (30) : (i + w + 1);

// for starting y coordinate to ending y coordinate
for (int j = start; j < end; j++)
           {
// compute the instantaneous cost for gestureA(i) to gestureB(j)
double cost = Math.Sqrt((gestureA[i, 0] - gestureB[j, 0]) * (gestureA[i, 0] - gestureB[j, 0]) +
                        (gestureA[i, 1] - gestureB[j, 1]) * (gestureA[i, 1] - gestureB[j, 1]) +
                        (gestureA[i, 2] - gestureB[j, 2]) * (gestureA[i, 2] - gestureB[j, 2]) +
                        (gestureA[i, 3] - gestureB[j, 3]) * (gestureA[i, 3] - gestureB[j, 3]));

// list the possible origins accumulated cost
double[] possibleOrigins = { DTWMatrix[i - 1, j], DTWMatrix[i, j - 1], DTWMatrix[i - 1, j - 1] };

// the actual accumulated cost is the minimal possible origin accumulated cost plus the instaneous cost
             DTWMatrix[i, j] = cost + possibleOrigins.Min();

           }
        }

// the DTW result may be the last element of the matrix or the 4 adjacent cells
double[] DTWresults = {  DTWMatrix[27, 29], DTWMatrix[28, 29], DTWMatrix[29, 29], DTWMatrix[29,
28], DTWMatrix[29, 27] };

// return the minimal of the possible DTW results
return DTWresults.Min();

     }
   }
}
```

19.6.6 Gesture.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Kinect;


namespace KinectAGVNavigator
{
class Gesture
  {
// list of sequences of the gesture
public List<GestureSequence> GestureSequences = new List<GestureSequence> { };

// index of the sequence position which is being modified
int GestureSequenceIndex = 0;

// gesture's name
public String GestureName;

/// <summary>
/// Constructor
/// </summary>
/// <param name="gn"> Gesture's name </param>
public Gesture(String gn)
    {
      GestureName = gn;
    }

/// <summary>
/// Stores angle array in new sequence
/// </summary>
/// <param name="angles"> Angles array to be stored in the sequence </param>
publicvoid newSequence(double[] angles)
    {
// if the index is 0, a new sequence has to be added
if (GestureSequenceIndex == 0)
      {

//add a new sequence
        GestureSequences.Add(new GestureSequence());
      }

// if the index is under 30
if (GestureSequenceIndex <30)
      {
// foreach feature
for (int i = 0; i <6; i++)
        {
```

```
//store it in the last position of the sequence
                GestureSequences.Last().Angles[GestureSequenceIndex, i] = angles[i];
            }

// increase sequence index
            GestureSequenceIndex++;

        }

    }

/// <summary>
/// Stops the recording of a new sequence, extract its means and normalize it
/// </summary>
publicvoid stopNewSequence()
    {
// reset sequence index
        GestureSequenceIndex = 0;

// create speace to allocate mean of each feature
double[] mean = newdouble[] { 0, 0, 0, 0, 0, 0 };

// for each frame of the sequence
for (int i = 0; i <30; i++)
        {
// sum of all frames for each feature
            mean[0] = mean[0] + GestureSequences.Last().Angles[i, 0];
            mean[1] = mean[1] + GestureSequences.Last().Angles[i, 1];
            mean[2] = mean[2] + GestureSequences.Last().Angles[i, 2];
            mean[3] = mean[3] + GestureSequences.Last().Angles[i, 3];
            mean[4] = mean[4] + GestureSequences.Last().Angles[i, 4];
            mean[5] = mean[5] + GestureSequences.Last().Angles[i, 5];
        }
// for each feature
for (int i = 0; i <6; i++)
        {
// divide sum by 30 frames obtaing the mean
            mean[i] = mean[i] / 30;

// store mean in the sequence's informations
            GestureSequences.Last().means[i] = mean[i];
        }

// for each frame in sequence
for (int i = 0; i <30; i++)
{
// substract mean to each feature (normalization)
            GestureSequences.Last().Angles[i, 0] = GestureSequences.Last().Angles[i, 0] - mean[0];
            GestureSequences.Last().Angles[i, 1] = GestureSequences.Last().Angles[i, 1] - mean[1];
            GestureSequences.Last().Angles[i, 2] = GestureSequences.Last().Angles[i, 2] - mean[2];
            GestureSequences.Last().Angles[i, 3] = GestureSequences.Last().Angles[i, 3] - mean[3];
            GestureSequences.Last().Angles[i, 4] = GestureSequences.Last().Angles[i, 4] - mean[4];
            GestureSequences.Last().Angles[i, 5] = GestureSequences.Last().Angles[i, 5] - mean[5];
```

```
        }
      }
    }
  }
```

19.6.7 GestureManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Kinect;
using System.IO; //to specify path
using SQLite; // for sqlite

namespace KinectAGVNavigator
{
class GestureManager
   {
// List of gestures
public List<Gesture> Gestures = new List<Gesture> { };


/// <summary>
/// GestureManager constructor
/// </summary>
public GestureManager()
      {
// do nothing in constructor
      }


/// <summary>
/// Add a new gesture
/// </summary>
/// <param name="title">title of the new gesture to be added</param>
publicvoid newGestureType(string title)
      {
// add gesture with the title "title" to the gesture list
         Gestures.Add(new Gesture(title));
      }

/// <summary>
/// Records a new gesture
/// </summary>
/// <param name="title">title of the gesture to which a new sequence has to be recorded</param>
/// <param name="angles">angles to be recorded to the gesture sequence</param>
publicvoid newSequence(string title, double[] angles)
      {
// finds the gesture with the title "title" and records the angles
        Gestures.Find(r => r.GestureName == title).newSequence(angles);
      }

/// <summary>
/// Stops to record a new gesture
/// </summary>
```

```
/// <param name="title">title of the new gesture</param>
publicvoid stopNewSequence(string title)
      {
// finds the gesture with the title "title" and stops the record
          Gestures.Find(r => r.GestureName == title).stopNewSequence();
      }

/// <summary>
/// Searchs for a gesture and returns if it already exists or not
/// </summary>
/// <param name="title">title of the new gesture to be searched</param>
publicbool GestureAllreadyExists(string title)
      {
// search for the gesture with title "title" and return true if found and false if not
return (Gestures.Exists(r => r.GestureName == title)) ? true: false;

}
   }
}
```

19.6.8 GestureSequence.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace KinectAGVNavigator
{
class GestureSequence
  {
// feature-fram matrix
publicdouble[,] Angles = newdouble[,]
{{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},

{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},

{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0},{0,0,0,0,0,0,0}};

// means array
publicdouble[] means = newdouble[] { 0, 0, 0, 0, 0, 0 };

}
}
```

19.6.9 NavigatorState.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace KinectAGVNavigator
{
class NavigatorState
  {
// velocity set point order for AGV
publicint speedSetPoint = 0;

// virtual macome sensor
publicint macome = 5634;

// detection dictionary
public Dictionary<string, bool> detection = new Dictionary<string, bool>();


/// <summary>
/// Generates a string with the status of the system to be sent to the AGV's processor
/// </summary>
publicstring getStatus()
    {
// BCD conversion of speed set point
int aux_speed = speedSetPoint;
int su = aux_speed % 10;
        aux_speed = aux_speed / 10;
int st = aux_speed % 10;
        aux_speed = aux_speed / 10;
int sh = aux_speed % 10;

// BCD conversion of virtual macome sensor
int aux_macome = macome;
int mu = aux_macome % 10;
        aux_macome = aux_macome / 10;
int mt = aux_macome % 10;
        aux_macome = aux_macome / 10;
int mh = aux_macome % 10;
        aux_macome = aux_macome / 10;
int mth = aux_macome % 10;
        aux_macome = aux_macome / 10;
int mtth = aux_macome % 10;

// generate void string
string status = "";

// speed's hundreds
        status += sh.ToString();
```

```
// speed's tens
        status += st.ToString();
// speed's units
        status += su.ToString();

// macome's ten thousands
        status += mtth.ToString();
// macome's thousands
        status += mth.ToString();
// macome's hundreds
        status += mh.ToString();
// macome's tens
        status += mt.ToString();
// macome's units
        status += mu.ToString();

// copy detection dictionary
        Dictionary<string, bool> detectionCopy = new Dictionary<string, bool>(detection);

// foreach entry in detectionCopy
foreach (KeyValuePair<string, bool> entry in detectionCopy)
        {
// write "1" if gesture was performed or "0" if not
            status += entry.Value ? "1" : "0";

// reset dictionary
            detection[entry.Key] = false;
        }

// return generated message
return status;

    }


/// <summary>
/// Computes new speed set point
/// </summary>
/// <param name="agvState"> AGV state </param>
/// <param name="userData"> User data </param>
publicvoid speedSetPointUpdate(AgvState agvState, UserData userData)
    {
// get user's speed relative to AGVs speed
double[] speed = userData.getVelocity();

// get distance to user
double[] distance = userData.getDistance();

// compute user's absolute speed
int userAbsoluteSpeed = agvState.speed + Convert.ToInt32( speed[1] * 100.0 / 1.1);

// compute the diference between the actual distance of the user and the desired distance
int distanceReference = Convert.ToInt32((2 - distance[2]) * 100);
```

```
// substrat distance reference from user's absolute speed
        speedSetPoint = userAbsoluteSpeed - distanceReference;

// if set point speed is larger than 100 set it to 100
if (speedSetPoint >100) speedSetPoint = 100;

// if set point speed is lower than 0 set it to 0
if (speedSetPoint <0) speedSetPoint = 0;
      }
   }
}
```

19.6.10 PlayerData.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;
using Microsoft.Kinect;

namespace KinectAGVNavigator
{
class PlayerData
  {
// space to alocate the last sequence performed by the player
public GestureSequence Sequence = new GestureSequence { };

// space to alocate the normalized sequence
public GestureSequence ACSequence = new GestureSequence { };

// Player identifier
publicint Player = 1;

// Player's body orientation towards the camera
double currentBodyOrientation = 0;

// Player's body's coordinates
double xBodyPosition = 0;
double yBodyPosition = 0;
double zBodyPosition = 0;

// Player's T-shirt's color
publicint[] color = newint[] { 0, 0, 0 };

/// <summary>
/// Constructor
/// </summary>
/// <param name="id"> Player's ID </param>
public PlayerData(int id)
    {
// store "id" in Players ID
        Player = id;
    }

/// <summary>
/// Check if the player is facing the camera
/// </summary>
publicbool isUserFacingToCamera()
    {
// return true if player is facing to camera, false if not
return (Math.Abs((Math.Atan2(-xBodyPosition, zBodyPosition) - currentBodyOrientation)) < Math.PI
/ 6) ? true : false;
```

```
    }

/// <summary>
/// Updates the information about the player
/// </summary>
/// <param name="PlayerBody"> Body information of the player </param>
publicvoid update(Body PlayerBody)
    {

// obtain angles from player's body info
double[] angles = getAngles(PlayerBody);

// update BodyPosition
        xBodyPosition = PlayerBody.Joints[JointType.SpineMid].Position.X;
        yBodyPosition = PlayerBody.Joints[JointType.SpineMid].Position.Y;
        zBodyPosition = PlayerBody.Joints[JointType.SpineMid].Position.Z;

// update current body Orientation
        currentBodyOrientation = angles[6];

// for each cell in the sequence matrix
for (int i = 0; i <29; i++)
        {
for (int j = 0; j <6; j++)
          {
// shift one position to the left
            Sequence.Angles[i, j] = Sequence.Angles[i + 1, j];
          }
        }

// for each most right feature
for (int j = 0; j <6; j++)
        {
// add new feature
          Sequence.Angles[29, j] = angles[j];
        }

// alocate speace for means
double[] means = newdouble[] { 0, 0, 0, 0, 0, 0 };

// for each frame in sequence
for (int i = 0; i <30; i++)
        {
// sum of all frames for each feature
          means[0] = means[0] + Sequence.Angles[i, 0];
          means[1] = means[1] + Sequence.Angles[i, 1];
          means[2] = means[2] + Sequence.Angles[i, 2];
          means[3] = means[3] + Sequence.Angles[i, 3];
          means[4] = means[4] + Sequence.Angles[i, 4];
          means[5] = means[5] + Sequence.Angles[i, 5];
        }
```

```
// for each feature
for (int i = 0; i <6; i++)
        {
// divide sum by 30 (mean)
            means[i] = means[i] / 30;

// store means
            Sequence.means[i] = means[i];
        }

// for each cell in ACsequence
for (int i = 0; i <30; i++)
        {
for (int j = 0; j <6; j++)
            {
// store the normalized sequence value
                ACSequence.Angles[i, j] = Sequence.Angles[i, j] - means[j];
            }
        }
    }

/// <summary>
/// Obtains the body angles
/// </summary>
/// <param name="PlayerBody"> Body information of the player </param>
privatedouble[] getAngles(Body PlayerBody)
    {

// obtain space position of joints
        CameraSpacePoint HandTip = PlayerBody.Joints[JointType.HandTipRight].Position;
        CameraSpacePoint Thumb = PlayerBody.Joints[JointType.ThumbRight].Position;
        CameraSpacePoint Wrist = PlayerBody.Joints[JointType.WristRight].Position;
        CameraSpacePoint Elbow = PlayerBody.Joints[JointType.ElbowRight].Position;
        CameraSpacePoint ShoulderR = PlayerBody.Joints[JointType.ShoulderRight].Position;
        CameraSpacePoint ShoulderL = PlayerBody.Joints[JointType.ShoulderLeft].Position;
        CameraSpacePoint Neck = PlayerBody.Joints[JointType.Neck].Position;

// body orientation code (selects the 3 optimal joints to compute body orientation)
int OrientationCode = 0;

// select optimal code to calculate Neck-Shoulder vector
if (PlayerBody.Joints[JointType.ShoulderRight].TrackingState == TrackingState.Tracked &&
PlayerBody.Joints[JointType.Neck].TrackingState == TrackingState.Tracked) OrientationCode = 12;
elseif (PlayerBody.Joints[JointType.ShoulderRight].TrackingState == TrackingState.Tracked &&
PlayerBody.Joints[JointType.ShoulderLeft].TrackingState == TrackingState.Tracked) OrientationCode
= 13;
elseif (PlayerBody.Joints[JointType.Neck].TrackingState == TrackingState.Tracked &&
PlayerBody.Joints[JointType.ShoulderLeft].TrackingState == TrackingState.Tracked) OrientationCode
= 23;
elseif (PlayerBody.Joints[JointType.ShoulderRight].TrackingState == TrackingState.Inferred &&
PlayerBody.Joints[JointType.Neck].TrackingState == TrackingState.Inferred) OrientationCode = 12;
```

```
elseif (PlayerBody.Joints[JointType.ShoulderRight].TrackingState == TrackingState.Inferred && PlayerBody.Joints[JointType.ShoulderLeft].TrackingState == TrackingState.Inferred) OrientationCode = 13;
elseif (PlayerBody.Joints[JointType.Neck].TrackingState == TrackingState.Inferred && PlayerBody.Joints[JointType.ShoulderLeft].TrackingState == TrackingState.Inferred) OrientationCode = 23;
else OrientationCode = 12;

// Neck-Shoulder vector in global reference
double xSN;
double ySN;
double zSN;

// select 3 points to obtain neck-shoulder vector in function of the orientation code
if (OrientationCode == 13)
        {
          xSN = ShoulderR.X - ShoulderL.X;
          ySN = ShoulderR.Y - ShoulderL.Y;
          zSN = ShoulderR.Z - ShoulderL.Z;
        }
elseif (OrientationCode == 23)
        {
          xSN = Neck.X - ShoulderL.X;
          ySN = Neck.Y - ShoulderL.Y;
          zSN = Neck.Z - ShoulderL.Z;
        }
else
        {
          xSN = ShoulderR.X - Neck.X;
          ySN = ShoulderR.Y - Neck.Y;
          zSN = ShoulderR.Z - Neck.Z;
        }

// Shoulder-Elbow vector in global reference

double xES = Elbow.X - ShoulderR.X;
double yES = Elbow.Y - ShoulderR.Y;
double zES = Elbow.Z - ShoulderR.Z;

// Elbow-Wrist vector in global reference

double xWE = Wrist.X - Elbow.X;
double yWE = Wrist.Y - Elbow.Y;
double zWE = Wrist.Z - Elbow.Z;

// Wrist-Thumb vector in global referece

double xTW = Thumb.X - Wrist.X;
double yTW = Thumb.Y - Wrist.Y;
double zTW = Thumb.Z - Wrist.Z;

// Wrist-Hand Tip vector in global reference
```

```
double xHW = HandTip.X - Wrist.X;
double yHW = HandTip.Y - Wrist.Y;
double zHW = HandTip.Z - Wrist.Z;

// Neck-Shoulder RM
double angleBody = Math.Atan2(zSN, xSN);
double[,] RMSN = newdouble[,] { { Math.Cos(angleBody), 0, -Math.Sin(angleBody) }, { 0, 1, 0 }, {
Math.Sin(angleBody), 0, Math.Cos(angleBody) } };

// Shoulder-Elbow vector in Neck-Shoulder reference (v' = R(^T)*v)

double xESrSN = RMSN[0, 0] * xES + RMSN[1, 0] * yES + RMSN[2, 0] * zES;
double yESrSN = RMSN[0, 1] * xES + RMSN[1, 1] * yES + RMSN[2, 1] * zES;
double zESrSN = RMSN[0, 2] * xES + RMSN[1, 2] * yES + RMSN[2, 2] * zES;

//RotationMatrix of Shoulder-Eblow

// x components of RotationMatrix of Shoulder-Elbow in global reference (directly normalized global
reference Shoulder-Elbow vector)

double xESeuclidean = Math.Sqrt(xES * xES + yES * yES + zES * zES);

double xxRMES = xES / xESeuclidean;
double xyRMES = yES / xESeuclidean;
double xzRMES = zES / xESeuclidean;

// z components of RotationMatrix of Shoulder-Elbow in global reference (normalized crossProduct of
global reference Shoulder-Elbow and Elbow-Wrist vectors)

double zxRMES = yES * zWE - zES * yWE;
double zyRMES = zES * xWE - xES * zWE;
double zzRMES = xES * yWE - yES * xWE;

double zESeuclidean = Math.Sqrt(zxRMES * zxRMES + zyRMES * zyRMES + zzRMES * zzRMES);

        zxRMES = zxRMES / zESeuclidean;
        zyRMES = zyRMES / zESeuclidean;
zzRMES = zzRMES / zESeuclidean;

// y components of RotationMatrix of Shoulder-Elbow in global reference (crossProduct of x and z
components of Shoulder-Eblow RM)

double yxRMES = xyRMES * zzRMES - xzRMES * zyRMES;
double yyRMES = xzRMES * zxRMES - xxRMES * zzRMES;
double yzRMES = xxRMES * zyRMES - xyRMES * zxRMES;

double[,] RMES = newdouble[,] { { xxRMES, yxRMES, zxRMES }, { xyRMES, yyRMES, zyRMES }, {
xzRMES, yzRMES, xzRMES } };

//RotationMatrix of Shoulder-Eblow without y component in x-ES y-SN plane (with fixes roll)

// x components
```

```
double xxRMESwFR = xxRMES;
double xyRMESwFR = xyRMES;
double xzRMESwFR = xzRMES;

//z components

double zxRMESwFR = -xzRMESwFR / Math.Sqrt(xxRMESwFR * xxRMESwFR + xzRMESwFR * xzRMESwFR);
double zyRMESwFR = 0;
double zzRMESwFR = xxRMESwFR / Math.Sqrt(xxRMESwFR * xxRMESwFR + xzRMESwFR * xzRMESwFR);

//y components

double yxRMESwFR = xyRMESwFR * zzRMESwFR - xzRMESwFR * zyRMESwFR;
double yyRMESwFR = xzRMESwFR * zxRMESwFR - xxRMESwFR * zzRMESwFR;
double yzRMESwFR = xxRMESwFR * zyRMESwFR - xyRMESwFR * zxRMESwFR;

double[,] RMESwFR = newdouble[,] { { xxRMES, yxRMESwFR, zxRMESwFR }, { xyRMES, yyRMESwFR, zyRMESwFR }, { xzRMES, yzRMESwFR, zzRMESwFR } };

//Shoulder-Elbow rotation matrix y components rotated in rotation matrix frame with fixed roll

double yxESrSN = RMESwFR[0, 0] * yxRMES + RMESwFR[1, 0] * yyRMES + RMESwFR[2, 0] * yzRMES;
double yyESrSN = RMESwFR[0, 1] * yxRMES + RMESwFR[1, 1] * yyRMES + RMESwFR[2, 1] * yzRMES;
double yzESrSN = RMESwFR[0, 2] * yxRMES + RMESwFR[1, 2] * yyRMES + RMESwFR[2, 2] * yzRMES;

//Elbow-Wrist in Shoulder-Elbow reference frame

double xWErES = RMES[0, 0] * xWE + RMES[1, 0] * yWE + RMES[2, 0] * zWE;
double yWErES = RMES[0, 1] * xWE + RMES[1, 1] * yWE + RMES[2, 1] * zWE;
double zWErES = RMES[0, 2] * xWE + RMES[1, 2] * yWE + RMES[2, 2] * zWE;

//Elbow-Wrist rotation matrix

double xRMWEeuclidean = Math.Sqrt(xWE * xWE + yWE * yWE + zWE * zWE);

// x column

double xxRMWE = xWE / xRMWEeuclidean;
double xyRMWE = yWE / xRMWEeuclidean;
double xzRMWE = zWE / xRMWEeuclidean;

// y column

double yxRMWE = yTW * xzRMWE - zTW * xyRMWE;
double yyRMWE = zTW * xxRMWE - xTW * xzRMWE;
double yzRMWE = xTW * xyRMWE - yTW * xxRMWE;

double yRMWEeuclidean = Math.Sqrt(yxRMWE * yxRMWE + yyRMWE * yyRMWE + yzRMWE * yzRMWE);

    yxRMWE = yxRMWE / yRMWEeuclidean;
```

```
        yyRMWE = yyRMWE / yRMWEeuclidean;
        yzRMWE = yzRMWE / yRMWEeuclidean;

// z column

double zxRMWE = xyRMWE * yzRMWE - xzRMWE * yyRMWE;
double zyRMWE = xzRMWE * yxRMWE - xxRMWE * yzRMWE;
double zzRMWE = xxRMWE * yyRMWE - xyRMWE * yxRMWE;

double[,] RMWE = newdouble[,] { { xxRMWE, yxRMWE, zxRMWE }, { xyRMWE, yyRMWE, zyRMWE }, {
xzRMWE, yzRMWE, zzRMWE } };

//Elbow-Wrist rotation matrix with fixed roll

// x column

double xxRMWEwFR = xxRMWE;
double xyRMWEwFR = xyRMWE;
double xzRMWEwFR = xzRMWE;

// z column

double zxRMWEwFR = yES * xzRMWEwFR - zES * xyRMWEwFR;
double zyRMWEwFR = zES * xxRMWEwFR - xES * xzRMWEwFR;
double zzRMWEwFR = xES * xyRMWEwFR - yES * xxRMWEwFR;

double zRMWEwFReuclidean = Math.Sqrt(zxRMWEwFR * zxRMWEwFR + zyRMWEwFR *
zyRMWEwFR + zzRMWEwFR * zzRMWEwFR);

        zxRMWEwFR = zxRMWEwFR / zRMWEwFReuclidean;
        zyRMWEwFR = zyRMWEwFR / zRMWEwFReuclidean;
        zzRMWEwFR = zzRMWEwFR / zRMWEwFReuclidean;

// y column

double yxRMWEwFR = zyRMWEwFR * xzRMWEwFR - zzRMWEwFR * xyRMWEwFR;
double yyRMWEwFR = zzRMWEwFR * xxRMWEwFR - zxRMWEwFR * xzRMWEwFR;
double yzRMWEwFR = zxRMWEwFR * xyRMWEwFR - zyRMWEwFR * xxRMWEwFR;

double[,] RMWEwFR = newdouble[,] { { xxRMWEwFR, yxRMWEwFR, zxRMWEwFR }, { xyRMWEwFR,
yyRMWEwFR, zyRMWEwFR }, { xzRMWEwFR, yzRMWEwFR, zzRMWEwFR } };

//Elbow-Wrist rotation matrix y components rotated in rotation matrix frame with fixed roll

double yxWErWEFR = RMWEwFR[0, 0] * zxRMWE + RMWEwFR[1, 0] * zyRMWE + RMWEwFR[2, 0] *
zzRMWE;
double yyWErWEFR = RMWEwFR[0, 1] * zxRMWE + RMWEwFR[1, 1] * zyRMWE + RMWEwFR[2, 1] *
zzRMWE;
double yzWErWEFR = RMWEwFR[0, 2] * zxRMWE + RMWEwFR[1, 2] * zyRMWE + RMWEwFR[2, 2] *
zzRMWE;

//Wrist-HandTip vector in Elbow-Wrist reference
```

```
double xHWrWE = RMWE[0, 0] * xHW + RMWE[1, 0] * yHW + RMWE[2, 0] * zHW;
double yHWrWE = RMWE[0, 1] * xHW + RMWE[1, 1] * yHW + RMWE[2, 1] * zHW;
double zHWrWE = RMWE[0, 2] * xHW + RMWE[1, 2] * yHW + RMWE[2, 2] * zHW;

// obtain each angle
double ShoulderYaw = Math.Atan2(zESrSN, xESrSN);
double ShoulderPitch = -Math.Atan2(yESrSN, Math.Sqrt(zESrSN * zESrSN + xESrSN * xESrSN));
double ESroll = Math.Atan2(yzESrSN, yyESrSN);
double pitchElbow = Math.Atan2(yWErES, xWErES);
double WEroll = Math.Atan2(yzWErWEFR, yyWErWEFR);
double angleHand = Math.Atan2(yHWrWE, xHWrWE);

// copy all angles into array
double[] angles = newdouble[] { ShoulderYaw, ShoulderPitch, ESroll, pitchElbow, WEroll, angleHand,
angleBody };

// return angles array
return angles;

    }


  }
}
```

19.6.11 UserData.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;
using Microsoft.Kinect;


namespace KinectAGVNavigator
{
class UserData
  {
// space to alocate the last sequence performed by the player
public GestureSequence Sequence = new GestureSequence { };

// space to alocate the normalized sequence
public GestureSequence ACSequence = new GestureSequence { };

// search for user flag
publicbool searchForUser = false;

// user is being tracked flag
publicbool isTracked = false;

// player id of the user
publicint PlayerID = 1;

// public body position
public CameraSpacePoint BodySpacePoint;

// timer to search user after being lost
    Stopwatch searchForPlayerTime = new Stopwatch();

// body orientation
double currentBodyOrientation = 0;

// body position
double xBodyPosition = 0;
double yBodyPosition = 0;
double zBodyPosition = 0;

// head position
publicdouble xHeadPosition = 0;
publicdouble yHeadPosition = 0;
publicdouble zHeadPosition = 0;

// user's color
publicint[] color = newint[] { 0, 0, 0 };

// timer for velocity calculation
```

```
    Stopwatch elapsedTime = new Stopwatch();

// body position in the last frame
double xlastBodyPosition = 0;
double zlastBodyPosition = 0;

// user's velocity
double xVel = 0;
double zVel = 0;

// frames counter
int frames = 0;


// posible user color information (color from 6 players)
int[] posibleUserHue = newint[] { 500, 500, 500, 500, 500, 500 };
int[] posibleUserSaturation = newint[] { 300, 300, 300, 300, 300, 300 };
int[] posibleUserValue = newint[] { 300, 300, 300, 300, 300, 300 };

/// <summary>
/// Constructor
/// </summary>
public UserData()
    {
      elapsedTime.Start();
    }


/// <summary>
/// updates color of players
/// </summary>
/// <param name="playerID"> playerID </param>
/// <param name="color"> color array </param>
publicvoid updatePosibleUserColor(int playerID, int[] color)
    {
//converts RGB color to HSV color
int[] hsvColor = RGBtoHSV(color);

// copy obtained color data
this.posibleUserHue[playerID] = hsvColor[0];
this.posibleUserSaturation[playerID] = hsvColor[1];
this.posibleUserValue[playerID] = hsvColor[2];
    }


/// <summary>
/// selects the user amonsts players
/// </summary>
/// <param name="trackingStates"> Tracking state array </param>
/// <param name="playerStarting"> Player starting flag array </param>
/// <param name="navState"> Navigation state </param>
publicvoid selectUser(bool[] trackingStates, bool[] playerStarting, NavigatorState navState)
    {
```

```
// if user in not being tracked and user is not being searched for
if (!this.isTracked && !this.searchForUser)
        {
// for each player
for (int i = 0; i <6; i++)
            {
// if player is tracked and is performing engage gesture
if (trackingStates[i] && playerStarting[i])
                {
// track this player as user
this.PlayerID = i;

// activate tracking flag
this.isTracked = true;

// break loop
break;
                }
            }
        }
// if user is being tracked
if (this.isTracked)
        {
// if the player of the user is not being tracked (user lost)
if (!trackingStates[this.PlayerID])
            {
// stop user tracking
this.isTracked = false;

// activate search for former user
this.searchForUser = true;

// start searching timer
                searchForPlayerTime.Start();

// send stop command to the AGV
                navState.detection["o_Stop"] = true;
            }
        }

// if user is being searched
if (this.searchForUser)
        {

// converts former user's RGB color to HSV color
int[] hsvColor = RGBtoHSV(this.color);

// define standard hue limit
int hueLimit = 20;

// erase hue limit if hue has no information
if ((hsvColor[1] <10&& hsvColor[2] <50) || (hsvColor[1] <20&& hsvColor[2] <20)  || hsvColor[2]
<15) hueLimit = 360;
```

```
// for each player
for (int i = 0; i <6; i++)
        {
// if the player's color is detected as the same as the former user's color
if (trackingStates[i] && (playerStarting[i] || ((Math.Abs(posibleUserHue[i] - hsvColor[0]) < hueLimit ||
(360 - Math.Abs(posibleUserHue[i] - hsvColor[0]) < hueLimit &&360 - Math.Abs(posibleUserHue[i] -
hsvColor[0]) >0)) && Math.Abs(posibleUserSaturation[i] - hsvColor[1]) <25&&
Math.Abs(posibleUserValue[i] - hsvColor[2]) <30)))
          {
// stop searching for user
this.searchForUser = false;

// reset timer
this.searchForPlayerTime.Reset();

// activate track user flag
this.isTracked = true;

// track user with the player's ID
this.PlayerID = i;

// break loop
break;
          }
        }

// if searching time has passed
if (searchForPlayerTime.ElapsedMilliseconds >15000)
        {
// reset timer
          searchForPlayerTime.Reset();

// stop searching for user
this.searchForUser = false;
        }
      }
    }


/// <summary>
/// Returns true if user is facing the camera, false if not
/// </summary>
publicbool isUserFacingToCamera()
    {
// Returns true if user is facing the camera, false if not
return (Math.Abs((Math.Atan2(-xBodyPosition, zBodyPosition) - currentBodyOrientation)) < Math.PI
/ 6) ? true : false;

    }


/// <summary>
```

```
/// Computes user's velocity respect to the AGV in x and y direction
/// </summary>
publicvoid computeVelocity()
    {
// increase frames count
      frames++;

// if frames count is 4
if (frames == 4)
      {
// if the elapsed time since the last check is lower than 200ms
if (elapsedTime.ElapsedMilliseconds <200)
        {
// compute velocity
          xVel = (xBodyPosition - xlastBodyPosition) / elapsedTime.ElapsedMilliseconds * 1000;
          zVel = (zBodyPosition - zlastBodyPosition) / elapsedTime.ElapsedMilliseconds * 1000;
        }
else// else, if time is greater than 200ms, it menas that there was probably an error
        {
// velocity equals 0
          xVel = 0.0;
          zVel = 0.0;
        }

// shift body position to "last body position"
        xlastBodyPosition = xBodyPosition;
        zlastBodyPosition = zBodyPosition;

// restart timer
        elapsedTime.Restart();

// restart frames counter
        frames = 0;
      }
    }


/// <summary>
/// Gets velocity as an array
/// </summary>
/// <param name=""></param>
publicdouble[] getVelocity()
    {
// copy velocity to array
double[] velocity = newdouble[] { xVel, zVel };

// return velocity array
return velocity;
    }


/// <summary>
/// Gets user's distance to the AGV
```

```
/// </summary>
/// <param name=""></param>
publicdouble[] getDistance()
    {
// copy distances to array
double[] distance = newdouble[] {xBodyPosition, yBodyPosition, zBodyPosition  };

// return distances array
return distance;
    }


/// <summary>
/// update user's information
/// </summary>
/// <param name="userBody"> Body information of the user </param>
publicvoid update(Body userBody)
    {

// obtain abgles
double[] angles = getAngles(userBody);

// update body position

        xBodyPosition = userBody.Joints[JointType.SpineMid].Position.X;
        yBodyPosition = userBody.Joints[JointType.SpineMid].Position.Y;
        zBodyPosition = userBody.Joints[JointType.SpineMid].Position.Z;

// update head position
        xHeadPosition = userBody.Joints[JointType.Head].Position.X;
        yHeadPosition = userBody.Joints[JointType.Head].Position.Y;
        zHeadPosition = userBody.Joints[JointType.Head].Position.Z;

// update public body position
        BodySpacePoint = userBody.Joints[JointType.SpineMid].Position;

// update current body Orientation
        currentBodyOrientation = angles[6];

// for each cell in sequence
for (int i = 0; i <29; i++)
        {
for (int j = 0; j <6; j++)
          {
// shift cell to the left
            Sequence.Angles[i, j] = Sequence.Angles[i+1, j];
          }
        }

// for each feature of sequence
for (int j = 0; j <6; j++)
        {
// add new value
```

```
            Sequence.Angles[29, j] = angles[j];
        }

// define means array
double[] means = newdouble[] { 0, 0, 0, 0, 0, 0 };

// for each frame in sequence
for (int i = 0; i <30; i++)
        {
// sum each feature
            means[0] = means[0] + Sequence.Angles[i, 0];
            means[1] = means[1] + Sequence.Angles[i, 1];
            means[2] = means[2] + Sequence.Angles[i, 2];
            means[3] = means[3] + Sequence.Angles[i, 3];
            means[4] = means[4] + Sequence.Angles[i, 4];
            means[5] = means[5] + Sequence.Angles[i, 5];
        }

// for each feature
for (int i = 0; i <6; i++)
        {
// divide sum by 30 to obtain mean
            means[i] = means[i] / 30;

// store mean
            Sequence.means[i] = means[i];
        }

// for each cell in sequence
for (int i = 0; i <30; i++)
        {
for (int j = 0; j <6; j++)
            {
// substract mean from sequence to obtain normalized value
                ACSequence.Angles[i, j] = Sequence.Angles[i, j] - means[j];
            }
        }
    }


/// <summary>
/// update virtual macome sensor
/// </summary>
/// <param name="navState"> Navigation status </param>
publicvoid updateMacome(NavigatorState navState){


// is user is not being tracked
if (!isTracked)
        {
// sensor deviation = 0
            navState.macome = 0;
        }
```

```
else
        {
//Distance from steering center to Macome sensor
int d = 193;

//Macome length
int l = 150;

// Mmax and Mmin depend on the used PLC:

//Siemens
int Mmax = 22080;
int Mmin = 5500;

//Rockwell
//int Mmax = 22080;
//int Mmid = 5500;

// compute virtual sensor's value
int Mmid = (Mmax + Mmin) / 2;
int Mx = Convert.ToInt32(Math.Atan2(xBodyPosition, zBodyPosition) * d);
int m = 2*(Mmax - Mmid) / l;
            navState.macome = Mmid + Mx * m;
        }
    }


/// <summary>
/// Obtain angle's array of user
/// </summary>
/// <param name="userBody"> user's body information </param>
privatedouble[] getAngles(Body userBody)
    {
// obtain space position of joints
        CameraSpacePoint HandTip = userBody.Joints[JointType.HandTipRight].Position;
        CameraSpacePoint Thumb = userBody.Joints[JointType.ThumbRight].Position;
        CameraSpacePoint Wrist = userBody.Joints[JointType.WristRight].Position;
        CameraSpacePoint Elbow = userBody.Joints[JointType.ElbowRight].Position;
        CameraSpacePoint ShoulderR = userBody.Joints[JointType.ShoulderRight].Position;
        CameraSpacePoint ShoulderL = userBody.Joints[JointType.ShoulderLeft].Position;
        CameraSpacePoint Neck = userBody.Joints[JointType.Neck].Position;

// body orientation code (selects the 3 optimal joints to compute body orientation)
int OrientationCode = 0;

// select optimal code to calculate Neck-Shoulder vector
if (userBody.Joints[JointType.ShoulderRight].TrackingState == TrackingState.Tracked &&
userBody.Joints[JointType.Neck].TrackingState == TrackingState.Tracked) OrientationCode = 12;
elseif (userBody.Joints[JointType.ShoulderRight].TrackingState == TrackingState.Tracked &&
userBody.Joints[JointType.ShoulderLeft].TrackingState == TrackingState.Tracked) OrientationCode =
13;
```

```
elseif (userBody.Joints[JointType.Neck].TrackingState == TrackingState.Tracked && userBody.Joints[JointType.ShoulderLeft].TrackingState == TrackingState.Tracked) OrientationCode = 23;
elseif (userBody.Joints[JointType.ShoulderRight].TrackingState == TrackingState.Inferred && userBody.Joints[JointType.Neck].TrackingState == TrackingState.Inferred) OrientationCode = 12;
elseif (userBody.Joints[JointType.ShoulderRight].TrackingState == TrackingState.Inferred && userBody.Joints[JointType.ShoulderLeft].TrackingState == TrackingState.Inferred) OrientationCode = 13;
elseif (userBody.Joints[JointType.Neck].TrackingState == TrackingState.Inferred && userBody.Joints[JointType.ShoulderLeft].TrackingState == TrackingState.Inferred) OrientationCode = 23;
else OrientationCode = 12;

// Neck-Shoulder vector in global reference
double xSN;
double ySN;
double zSN;

// select 3 points to obtain neck-shoulder vector in function of the orientation code
if (OrientationCode == 13)
        {
          xSN = ShoulderR.X - ShoulderL.X;
          ySN = ShoulderR.Y - ShoulderL.Y;
          zSN = ShoulderR.Z - ShoulderL.Z;
        }
elseif (OrientationCode == 23)
        {
          xSN = Neck.X - ShoulderL.X;
          ySN = Neck.Y - ShoulderL.Y;
          zSN = Neck.Z - ShoulderL.Z;
        }
else
        {
          xSN = ShoulderR.X - Neck.X;
          ySN = ShoulderR.Y - Neck.Y;
          zSN = ShoulderR.Z - Neck.Z;
        }

// Shoulder-Elbow vector in global reference

double xES = Elbow.X - ShoulderR.X;
double yES = Elbow.Y - ShoulderR.Y;
double zES = Elbow.Z - ShoulderR.Z;

// Elbow-Wrist vector in global reference

double xWE = Wrist.X - Elbow.X;
double yWE = Wrist.Y - Elbow.Y;
double zWE = Wrist.Z - Elbow.Z;

// Wrist-Thumb vector in global referece

double xTW = Thumb.X - Wrist.X;
```

```
double yTW = Thumb.Y - Wrist.Y;
double zTW = Thumb.Z - Wrist.Z;

// Wrist-Hand Tip vector in global reference

double xHW = HandTip.X - Wrist.X;
double yHW = HandTip.Y - Wrist.Y;
double zHW = HandTip.Z - Wrist.Z;

// Neck-Shoulder RM
double angleBody = Math.Atan2(zSN, xSN);
double[,] RMSN = newdouble[,] { { Math.Cos(angleBody), 0, -Math.Sin(angleBody) }, { 0, 1, 0 }, { Math.Sin(angleBody), 0, Math.Cos(angleBody) } };

// Shoulder-Elbow vector in Neck-Shoulder reference (v' = R(^T)*v)

double xESrSN = RMSN[0, 0] * xES + RMSN[1, 0] * yES + RMSN[2, 0] * zES;
double yESrSN = RMSN[0, 1] * xES + RMSN[1, 1] * yES + RMSN[2, 1] * zES;
double zESrSN = RMSN[0, 2] * xES + RMSN[1, 2] * yES + RMSN[2, 2] * zES;

//RotationMatrix of Shoulder-Eblow

// x components of RotationMatrix of Shoulder-Elbow in global reference (directly normalized global reference Shoulder-Elbow vector)

double xESeuclidean = Math.Sqrt(xES * xES + yES * yES + zES * zES);

double xxRMES = xES / xESeuclidean;
double xyRMES = yES / xESeuclidean;
double xzRMES = zES / xESeuclidean;

// z components of RotationMatrix of Shoulder-Elbow in global reference (normalized crossProduct of global reference Shoulder-Elbow and Elbow-Wrist vectors)

double zxRMES = yES * zWE - zES * yWE;
double zyRMES = zES * xWE - xES * zWE;
double zzRMES = xES * yWE - yES * xWE;

double zESeuclidean = Math.Sqrt(zxRMES * zxRMES + zyRMES * zyRMES + zzRMES * zzRMES);

    zxRMES = zxRMES / zESeuclidean;
    zyRMES = zyRMES / zESeuclidean;
    zzRMES = zzRMES / zESeuclidean;

// y components of RotationMatrix of Shoulder-Elbow in global reference (crossProduct of x and z components of Shoulder-Eblow RM)

double yxRMES = xyRMES * zzRMES - xzRMES * zyRMES;
double yyRMES = xzRMES * zxRMES - xxRMES * zzRMES;
double yzRMES = xxRMES * zyRMES - xyRMES * zxRMES;

double[,] RMES = newdouble[,] { { xxRMES, yxRMES, zxRMES }, { xyRMES, yyRMES, zyRMES }, { xzRMES, yzRMES, xzRMES } };
```

```
//RotationMatrix of Shoulder-Eblow with y component in x-ES y-SN plane (with fixed roll)

// x components

double xxRMESwFR = xxRMES;
double xyRMESwFR = xyRMES;
double xzRMESwFR = xzRMES;

//z components

double zxRMESwFR = -xzRMESwFR / Math.Sqrt(xxRMESwFR * xxRMESwFR + xzRMESwFR * xzRMESwFR);
double zyRMESwFR = 0;
double zzRMESwFR = xxRMESwFR / Math.Sqrt(xxRMESwFR * xxRMESwFR + xzRMESwFR * xzRMESwFR);

//y components

double yxRMESwFR = xyRMESwFR * zzRMESwFR - xzRMESwFR * zyRMESwFR;
double yyRMESwFR = xzRMESwFR * zxRMESwFR - xxRMESwFR * zzRMESwFR;
double yzRMESwFR = xxRMESwFR * zyRMESwFR - xyRMESwFR * zxRMESwFR;

double[,] RMESwFR = newdouble[,] { { xxRMES, yxRMESwFR, zxRMESwFR }, { xyRMES, yyRMESwFR, zyRMESwFR }, { xzRMES, yzRMESwFR, zzRMESwFR } };



//Shoulder-Elbow rotation matrix y components rotated in rotation matrix frame with fixed roll

double yxESrSN = RMESwFR[0, 0] * yxRMES + RMESwFR[1, 0] * yyRMES + RMESwFR[2, 0] * yzRMES;
double yyESrSN = RMESwFR[0, 1] * yxRMES + RMESwFR[1, 1] * yyRMES + RMESwFR[2, 1] * yzRMES;
double yzESrSN = RMESwFR[0, 2] * yxRMES + RMESwFR[1, 2] * yyRMES + RMESwFR[2, 2] * yzRMES;


//Elbow-Wrist in Shoulder-Elbow reference frame

double xWErES = RMES[0, 0] * xWE + RMES[1, 0] * yWE + RMES[2, 0] * zWE;
double yWErES = RMES[0, 1] * xWE + RMES[1, 1] * yWE + RMES[2, 1] * zWE;
double zWErES = RMES[0, 2] * xWE + RMES[1, 2] * yWE + RMES[2, 2] * zWE;

//Elbow-Wrist rotation matrix

double xRMWEeuclidean = Math.Sqrt(xWE * xWE + yWE * yWE + zWE * zWE);

// x column

double xxRMWE = xWE / xRMWEeuclidean;
double xyRMWE = yWE / xRMWEeuclidean;
double xzRMWE = zWE / xRMWEeuclidean;

// y column
```

```
double yxRMWE = yTW * xzRMWE - zTW * xyRMWE;
double yyRMWE = zTW * xxRMWE - xTW * xzRMWE;
double yzRMWE = xTW * xyRMWE - yTW * xxRMWE;

double yRMWEeuclidean = Math.Sqrt(yxRMWE * yxRMWE + yyRMWE * yyRMWE + yzRMWE * yzRMWE);

    yxRMWE = yxRMWE / yRMWEeuclidean;
    yyRMWE = yyRMWE / yRMWEeuclidean;
    yzRMWE = yzRMWE / yRMWEeuclidean;

// z column

double zxRMWE = xyRMWE * yzRMWE - xzRMWE * yyRMWE;
double zyRMWE = xzRMWE * yxRMWE - xxRMWE * yzRMWE;
double zzRMWE = xxRMWE * yyRMWE - xyRMWE * yxRMWE;

double[,] RMWE = newdouble[,] { { xxRMWE, yxRMWE, zxRMWE }, { xyRMWE, yyRMWE, zyRMWE }, { xzRMWE, yzRMWE, zzRMWE } };

//Elbow-Wrist rotation matrix with fixed roll

// x column

double xxRMWEwFR = xxRMWE;
double xyRMWEwFR = xyRMWE;
double xzRMWEwFR = xzRMWE;

// z column

double zxRMWEwFR = yES * xzRMWEwFR - zES * xyRMWEwFR;
double zyRMWEwFR = zES * xxRMWEwFR - xES * xzRMWEwFR;
double zzRMWEwFR = xES * xyRMWEwFR - yES * xxRMWEwFR;

double zRMWEwFReuclidean = Math.Sqrt(zxRMWEwFR * zxRMWEwFR + zyRMWEwFR * zyRMWEwFR + zzRMWEwFR * zzRMWEwFR);

    zxRMWEwFR = zxRMWEwFR / zRMWEwFReuclidean;
    zyRMWEwFR = zyRMWEwFR / zRMWEwFReuclidean;
    zzRMWEwFR = zzRMWEwFR / zRMWEwFReuclidean;

// y column

double yxRMWEwFR = zyRMWEwFR * xzRMWEwFR - zzRMWEwFR * xyRMWEwFR;
double yyRMWEwFR = zzRMWEwFR * xxRMWEwFR - zxRMWEwFR * xzRMWEwFR;
double yzRMWEwFR = zxRMWEwFR * xyRMWEwFR - zyRMWEwFR * xxRMWEwFR;

double[,] RMWEwFR = newdouble[,] { { xxRMWEwFR, yxRMWEwFR, zxRMWEwFR }, { xyRMWEwFR, yyRMWEwFR, zyRMWEwFR }, { xzRMWEwFR, yzRMWEwFR, zzRMWEwFR } };

//Elbow-Wrist rotation matrix y components rotated in rotation matrix frame with fixed roll
```

```
double yxWErWEFR = RMWEwFR[0, 0] * zxRMWE + RMWEwFR[1, 0] * zyRMWE + RMWEwFR[2, 0] *
zzRMWE;
double yyWErWEFR = RMWEwFR[0, 1] * zxRMWE + RMWEwFR[1, 1] * zyRMWE + RMWEwFR[2, 1] *
zzRMWE;
double yzWErWEFR = RMWEwFR[0, 2] * zxRMWE + RMWEwFR[1, 2] * zyRMWE + RMWEwFR[2, 2] *
zzRMWE;

//Wrist-HandTip vector in Elbow-Wrist reference

double xHWrWE = RMWE[0, 0] * xHW + RMWE[1, 0] * yHW + RMWE[2, 0] * zHW;
double yHWrWE = RMWE[0, 1] * xHW + RMWE[1, 1] * yHW + RMWE[2, 1] * zHW;
double zHWrWE = RMWE[0, 2] * xHW + RMWE[1, 2] * yHW + RMWE[2, 2] * zHW;

// obtain each angle
double ShoulderYaw = Math.Atan2(zESrSN, xESrSN);
double ShoulderPitch = -Math.Atan2(yESrSN, Math.Sqrt(zESrSN * zESrSN + xESrSN * xESrSN));
double ESroll = Math.Atan2(yzESrSN, yyESrSN);
double pitchElbow = Math.Atan2(yWErES, xWErES);
double WEroll = Math.Atan2(yzWErWEFR, yyWErWEFR);
double angleHand = Math.Atan2(yHWrWE, xHWrWE);

// copy all angles into array
double[] angles = newdouble[] { ShoulderYaw, ShoulderPitch, ESroll, pitchElbow, WEroll, angleHand,
angleBody };

// return angles array
return angles;

    }


/// <summary>
/// Convert RGB color to HSV
/// </summary>
/// <param name="rgbColor"> RGB color to be converted </param>
privateint[] RGBtoHSV(int[] rgbColor)
    {
// normalize RGB color
double red = rgbColor[0] / 255.0;
double green = rgbColor[1] / 255.0;
double blue = rgbColor[2] / 255.0;

// defines support parameters
double max;
double min;
string maxC;

// if red value is higher than green and blue
if (red > green && red > blue)
        {
// red has the greatest intensity
            max = red;
            maxC = "r";
```

```
// if green value is lower than blue
if (green < blue)
        {
// green has the lowest intesity
min = green;
        }
else
        {
// blue has the lowest intesity
          min = blue;
        }
      }
// else if green value is higher than red and blue
elseif (green > blue && green > red)
      {
// green has the greatest intesity
        max = green;
        maxC = "g";

// if blue has a lower value than red
if (blue < red)
        {
// blue has the lowest intensity
          min = blue;
        }
else
        {
// red has the lowest intesity
          min = red;
        }
      }
else// else if blue has the greatest intensity
      {
// blue has the greatest intensity
        max = blue;
        maxC = "b";

// if green value is lower than red
if (green < red)
        {
// green has the lowest intesity
          min = green;
        }
else
        {
// red has the lowest intesity
          min = red;
        }
      }

// obtain Value
int Value = Convert.ToInt16(max * 100);
```

```
// define saturation and hue
int Saturation;
int Hue;

// if maximal intensity is 0, the color is black
if (max == 0)
        {
// hue is 0
            Hue = 0;

// saturation is 0
            Saturation = 0;
        }
else
        {
// compute saturation
            Saturation = Convert.ToInt16((1 - (min / max)) * 100);

// if max and min are the same (or almost the same)
if (max -  min <0.004)
            {
// Hue = 0, gray tone
                Hue = 0;
            }
// else if red has the gratest intensity
elseif (maxC == "r")
            {
// if green is higher than blue
if (green >= blue)
                {
// compute hue
                    Hue = Convert.ToInt16(60 * (green - blue) / (max - min));
                }
else
                {
// compute hue
                    Hue = Convert.ToInt16(60 * (6 + ((green - blue) / (max - min))));
                }
            }  //else if green has the greatest intesity
elseif (maxC == "g")
            {
// compute hue
                Hue = Convert.ToInt16(60 * (2 + (blue - red) / (max - min)));
            }
else
            {
// compute hue
                Hue = Convert.ToInt16(60 * (4 + (red - green) / (max - min)));
            }
        }

// copy hue saturation and value into HSL color array
```

```
int[] hslColor = newint[] { Hue, Saturation, Value };

// return HSL color array
return hslColor;
      }
   }
}
```

19.6.12 VirtualAssistantState.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;

namespace KinectAGVNavigator
{
class VirtualAssistantState
   {
// left eye x position
int leftEyeX = 180;

// left eye y position
int leftEyeY = 254;

// right eye x position
int rightEyeX = 380;

// right eye y position
int rightEyeY = 254;

// counter
int searchCounter = 0;

// counter limit
double maxCounter = 250.0;

//74 92 113

/// <summary>
/// Updates eye's position
/// </summary>
/// <param name="agvState"> agv status </param>
/// <param name="navState"> navigation status </param>
/// <param name="user"> user data </param>
/// <param name="trackingStates"> tracking states array</param>
/// <param name="Face"> Grid in which the face is drawn </param>
publicint[] update(AgvState agvState, NavigatorState navState, UserData user, bool[] trackingStates,
Grid Face)
      {
// eyeposition array
int[] eyePosition = newint[] { leftEyeX, leftEyeY, rightEyeX, rightEyeY };

// if AGV is in alarm state
if (agvState.alarm)
        {
```

```
// create a new image brush
            ImageBrush Brush = new ImageBrush();

// create a new image
            Image image = new Image();

// source alert image
            image.Source = new BitmapImage(new Uri(BaseUriHelper.GetBaseUri(Face),
"Images/alert.png"));

// source image brush
            Brush.ImageSource = image.Source;

// paint background with brush
            Face.Background = Brush;

        }
// if AGV is in warning state
elseif (agvState.warning)
        {
// create a new image brush
            ImageBrush Brush = new ImageBrush();

// create a new image
            Image image = new Image();

// source alert image
            image.Source = new BitmapImage(new Uri(BaseUriHelper.GetBaseUri(Face),
"Images/alert.png"));

// source image brush
            Brush.ImageSource = image.Source;

// paint background with brush
            Face.Background = Brush;

        } // else, if user is not being tracked
elseif (!user.isTracked)
        {
// at least 1 of the 6 players is being tracked flag
bool anyPlayerTracked = false;

// for each player
foreach (bool trackingState in trackingStates)
            {
// if tracked
if (trackingState)
                {
// activate flag
                    anyPlayerTracked = true;

// break loop
break;
```

```
            }
        }
if (user.searchForUser)
        {
// create a new image brush
            ImageBrush Brush = new ImageBrush();

// create a new image
            Image image = new Image();

// source lost image
            image.Source = new BitmapImage(new Uri(BaseUriHelper.GetBaseUri(Face),
"Images/lost.png"));

// source image brush
            Brush.ImageSource = image.Source;

// paint background with brush
            Face.Background = Brush;

// update eyeposition (eyes roll)
            eyePosition[0] = eyePosition[0] + Convert.ToInt32(25.0 * Math.Sin(searchCounter /
maxCounter * 2.0 * Math.PI));
            eyePosition[1] = eyePosition[1] - Convert.ToInt32(5.0 * Math.Cos(searchCounter /
maxCounter * 2.0 * Math.PI));
            eyePosition[2] = eyePosition[2] + Convert.ToInt32(25.0 * Math.Sin(searchCounter /
maxCounter * 2.0 * Math.PI));
            eyePosition[3] = eyePosition[3] - Convert.ToInt32(5.0 * Math.Cos(searchCounter /
maxCounter * 2.0 * Math.PI));

// increase search counter
            searchCounter++;

// if search counter is greater than maxCounter reset it
if (searchCounter > maxCounter) searchCounter = 0;


        }
// else if no player is being tracked
elseif (!anyPlayerTracked)
        {
// create a new image brush
            ImageBrush Brush = new ImageBrush();

// create a new image
            Image image = new Image();

// source sleeping image
            image.Source = new BitmapImage(new Uri(BaseUriHelper.GetBaseUri(Face),
"Images/Sleeping.png"));

// source image brush
            Brush.ImageSource = image.Source;
```

```
// paint background with brush
            Face.Background = Brush;
        }
else
        {
// create a new image brush
            ImageBrush Brush = new ImageBrush();

// create a new image
            Image image = new Image();

// source searching image
            image.Source = new BitmapImage(new Uri(BaseUriHelper.GetBaseUri(Face),
"Images/Searching.png"));

// source image brush
            Brush.ImageSource = image.Source;

// paint background with brush
            Face.Background = Brush;

// update eyeposition (eyes roll)
            eyePosition[0] = eyePosition[0] + Convert.ToInt32(25.0 * Math.Sin(searchCounter /
maxCounter * 2.0 * Math.PI));
            eyePosition[1] = eyePosition[1] - Convert.ToInt32(5.0 * Math.Cos(searchCounter /
maxCounter * 2.0 * Math.PI));
            eyePosition[2] = eyePosition[2] + Convert.ToInt32(25.0 * Math.Sin(searchCounter /
maxCounter * 2.0 * Math.PI));
            eyePosition[3] = eyePosition[3] - Convert.ToInt32(5.0 * Math.Cos(searchCounter /
maxCounter * 2.0 * Math.PI));

// increase search counter
            searchCounter++;

// if search counter is greater than maxCounter reset it
if (searchCounter > maxCounter) searchCounter = 0;
        }
    } // else if user is being tracked
elseif (user.isTracked)
    {
// if AGV is stopped
if (!agvState.run)
        {
// create a new image brush
            ImageBrush Brush = new ImageBrush();

// create a new image
            Image image = new Image();

// source searching image
            image.Source = new BitmapImage(new Uri(BaseUriHelper.GetBaseUri(Face),
"Images/Searching.png"));
```

```
// source image brush
            Brush.ImageSource = image.Source;

// paint background with brush
            Face.Background = Brush;
        }
else// else if AGV is running
        {
// create a new image brush
            ImageBrush Brush = new ImageBrush();

// create a new image
            Image image = new Image();

// source running image
            image.Source = new BitmapImage(new Uri(BaseUriHelper.GetBaseUri(Face),
"Images/Running.png"));

// source image brush
            Brush.ImageSource = image.Source;

// paint background with brush
            Face.Background = Brush;
        }

// get heads position
double x = user.xHeadPosition;
double y = user.yHeadPosition;
double z = user.zHeadPosition;

// compute angles to head
double horAngle = Math.Atan2(x, z);
double verAngle = Math.Atan2(y, z);

// rescale angles
double eyeXD = 50 * 3 * horAngle / Math.PI;
double eyeYD = 32 * 3 * verAngle / Math.PI;

// updates eyes position to look to the head
            eyePosition[0] = eyePosition[0] + Convert.ToInt32(eyeXD);
            eyePosition[1] = eyePosition[1] - Convert.ToInt32(eyeYD);
            eyePosition[2] = eyePosition[2] + Convert.ToInt32(eyeXD);
            eyePosition[3] = eyePosition[3] - Convert.ToInt32(eyeYD);

        }
// return eyes position array
return eyePosition;
}
  }
}
```