CRANFIELD UNIVERSITY

Achille MARTIN
Yang YOU
Chirantan SAHASRABUDHE

In-to-net Bot (ITNB)

Ball catching robot based on Control approach and enhanced with Reinforcement Learning methods

SATM
Robotics

Group Thesis
Academic Year: 2018 - 2019

Supervisors: Dr. Gilbert Tang
Associate Supervisor: Dr. Antonios Antoniadis
May 2019

CRANFIELD UNIVERSITY

SATM
Robotics

Group Thesis

Academic Year 2018 - 2019

Achille MARTIN
Yang YOU
Chirantan SAHASRABUDHE

In-to-net Bot (ITNB)

Ball catching robot based on Control approach and enhanced with Reinforcement Learning methods

Supervisor: Dr. Gilbert Tang
Associate Supervisor: Dr. Antonios Antoniadis
May 2019

ABSTRACT

This group thesis aims to explore the “In-to-Net Bot” system designed to explore “catching a ball” with a robot. It aims to contribute to solving logistics and low mobility tasks. The integration of the Dobot Magician (manipulator robot), the Microsoft Kinect, OpenCV software library, and ROS interfaces, are thoroughly explained for the implementation of a physical model. The integration of the Gazebo simulation environment, Rviz and Moveit! frameworks, and ROS nodes are also detailed for the implementation of a virtual model enhanced with reinforcement learning. The main coding languages are C++ and python.

A successful physical catching robot has been designed to catch dropped balls. A satisfactory positioning error of 5.1787±1.3081 mm (95% confidence interval) with the manipulator has been reached. The average frame rate per second on obtained didn’t meet the expectations on the other hand, as it reached only 7 to 10 frames per second (instead of 50 fps announced by Microsoft for Kinect version 2). Moreover, the movement of the robot arm is slightly slower than the ball motion after it has been thrown and takes up to 1.5 seconds in average. Consequently, the catch rate over 100 ball drops has been evaluated at 15%.

Keywords:

Robotics – Control theory – Reinforcement learning – Robot Vision

ACKNOWLEDGEMENTS

We would like to express our very great appreciation to Dr. Tang and Dr. Antoniadis for their valuable and constructive suggestions throughout the project.

Our special thanks are extended to the members of the Robotics Laboratory, who provided necessary equipment and helped us setup the experiments.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

IT	Information Technology
ITNB	In-to-net Bot

1 Introduction

People with less mobility are required to throw things for them to reach a place. A system that helps with this can be very useful. A robot that can be useful within a workshop to collect nuts and bolts or throw rubbish away as the worker stays in place. On a larger scale this could help with for fruits on high trees which may be dangerous when falling to the ground or are required not to be damaged e.g. Coconuts are thrown, felled or naturally fall from high heights, where there is always a risk of a free-falling coconut hitting an animal or person.

The main objective was: To demonstrate a concept by designing a system for the Dobot Magician to “catch” a ball. The team consisted of 3 novice MSc Robotics students with only approx. 75 days to come up with a solution.

1.1 Concept of ball catching

The concept of ball catching has been a topic for both robotics and neuroscience in regards with a human or animal being able to catch an object. Even the ability to “catch” or grip on to a static object such as a ledge or tree branches is included in such research. Research in to the learning that occurs by failing first, such as a baby learning to walk or to eventually, when it becomes a heuristic of the human, where they don’t have to put effort or thought in to walking.

The most recent paper (Doctoral Thesis – 2017) addressing the machine learning problem associated with the ball catching task is [1]. The team developed the abstract aspect of machine learning and the concrete one to help engineers design more effective solutions to Reinforcement Learning (RL) problems. The “spectrum of decomposability” is introduced to “characterise problems and solutions in decision making”. It consists in dividing a problem into several easier subproblems to solve it.

Concerning the ball catching problem, they compare the heuristics approach to the optimisation-based approach, as shown in Table 1.1. The result is that the best approach heavily depends on the assumptions made.

Table 1.1 - Comparison heuristics and optimisation approaches

Method	Heuristics	Optimisation based
Type	Specialist	Generalist
Definition	"Humans employ heuristics to reactively choose appropriate actions based on immediate visual feedback" [2] Practical method, not optimal	"Predict the ball trajectory to plan future actions" [2]
Input representation	Angular representation of the ball position (easily extracted from a camera sensor) with respect to the agent	Cartesian representation
Setup	Model-free approach	Model-based approach
Advocates	Chapman [3]	Belousov [2]
Strategy	Linear optical trajectory (LOT) Based on the rise of the tangent angle θ between ground plane and the ball at constant rate. An illustration is provided in Figure 1.1.	Linear quadratic Gaussian control (LQG) or Model Predictive control (MPC) Predict the impact point of the ball using the ball dynamics
Advantage	Performs well with systematic perturbations (air resistance)	Adapts well to Gaussian noise (Bayesian filtering)
Drawback	Harder to transfer to artificial systems	Generally applied in closed-room environments (not baseball stadiums)

The ball catching task is defined as follows: “build a robot that runs such that it intercepts a baseball flying high up in the air before it hits the ground”. They start by setting up the RL problem and addressing common issues in RL such as the “representation learning”, “curse of dimensionality”, “temporal credit assignment”, exploration vs. exploitation dilemma”.

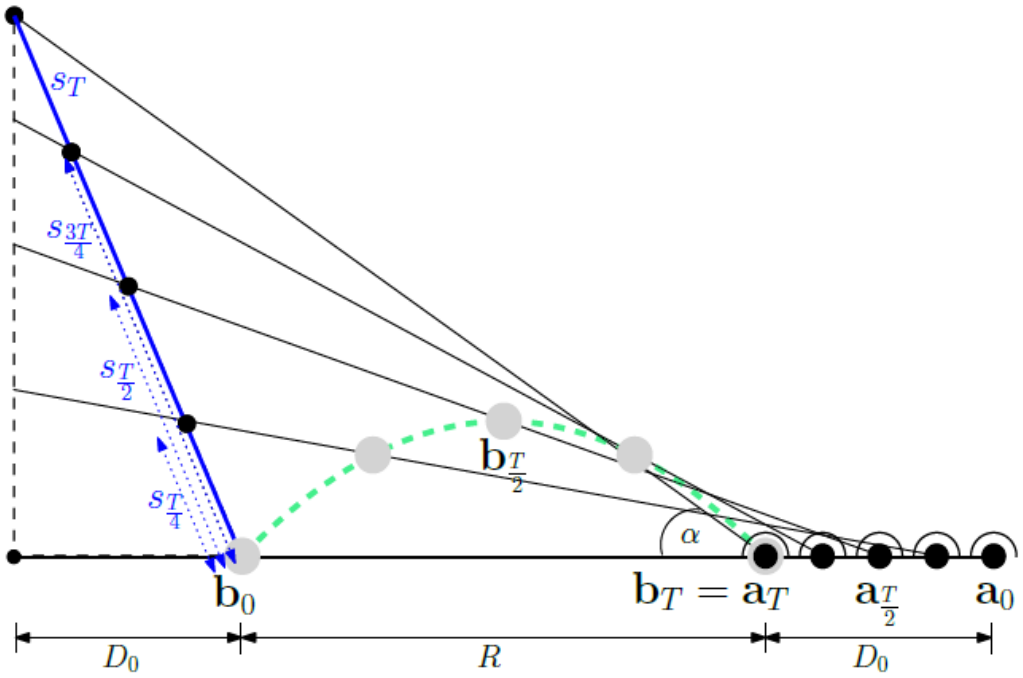


Figure 1.1 - Linear optical trajectory from Chapman

1.2 Scope and objectives

The model proposed to study the catching problem contains:

- Dynamics of the ball
- 2D movement of the agent
- Task (minimisation problem)
- Angular representation and cartesian representation

Note that the problem set up is slightly more complicated in the paper mentioned previously [1], since the robot is running backwards and may have to run forward without seeing the ball. In our robot catcher problem, an assumption can be made, stating that the robot stands still and can see the ball at any time.

The first objective of the study is to effectively convert ball motion into a simple entity relative to the robot/viewer. According to Table 1.1, on the one hand, Chapman proposes an angular representation based on the rise of the tangent angle θ between ground plane and the ball. On the other hand, Belousov advocates a cartesian representation and a prediction of the landing point of the ball using the dynamics and optimisation theory. This representation is easily applied to robots thanks to control theory and recent robotics advances. On the contrary, Chapman's representation is better suited to humans. Therefore, the control theory approach and the cartesian representation have been considered to tackle the issue of representing the ball from the robot point of vue.

The second objective resulting from the representation consideration is to use the ball landing point estimation to move the robot end-effector to this point. It concerns the control of the robot in conjunction with ball motion representation.

The third dimension of the study is to extend the control perspective of the robot. The RL problem stated in [1] is addressed in a 2D space (lower-dimensional camera). The camera model, the controller and the biases for learning (Linear P-control policy, policy search using CMA-ES for instance) are introduced. Results of the RL experiment show that "it is possible to learn a successful ball catching policy directly on observations, using generic reinforcement learning" [1].

Moreover, the policies learned are like the angular controllers (Heuristic method). However, since the result depends on the choice of the approach (model-free or model-based), a hybrid approach is possible.

Since the machine learning seems to be successfully applied to ball catching, the third objective of the study is then to consider reinforcement learning to improve the control of the robot to increase the catching rate. It is also to confirm whether reinforcement learning can play a significant role in ball catching for future research.

2 Literature review

2.1 Historical

To begin with, early research in robotics control, computer vision, artificial intelligence, machine learning, and mathematical techniques have been investigated. We saw that all these early research projects form the basis of the current research of today which they have gone on to build on top of.

As we wished to investigate machine learning methods such as reinforcement learning we looked in to the Q learning algorithm. In 1989, Watkins proposed Q learning algorithm in his PhD thesis [4] which is one of the most important algorithms in reinforcement learning for discretized setting.

In 1994 Peter Corke in his PhD [5] demonstrated the ability to track an object or ball using a manipulator and camera system.

A study conducted in 1994 by Das [6] on fly ball catching showed the use of machine learning / RL to solve the problem. They investigate whether the result from experimental psychology (i.e. Chapman) is enough for an agent to learn the task. A single camera is also used in a mousetrap catching system in Buttazzo et al. [7]. This is similarly tested for a manipulator testbed for catching objects by Fernandez et al.[8].

Back in 1995, the famous 4DOF “WAM” arm [9] at the MIT was created and was able to catch a ball using active vision system. The object was located and tracked, then the camera and the manipulator were calibrated (common coordinated frame), then the path was generated by determining the catch point and finally the arm would move to the designated point in time.

In 1997 the robot “Saika” developed by Nishiwaki [10] was able to localise a falling ball using 2 DDC cameras, predict the ball path and position its hand/basket at the catching point using inverse kinematics neural network method (three layered neural network with sigmoid function used).

2.2 Renowned research

As we forayed further in to current methods we saw that there are several simulation environments available for our research area. These include environments such as Gym, Mujoco, Gazebo, Bullet, Roboshool etc.

In the hope of going further with the machine learning approach, a simple “game of catch” [11] has been created by the Google DeepMind team to learn a control policy in a dynamic visual environment. The game is simple and involves only two objects: “a single pixel that represents a ball falling from the top of the screen while bouncing off the sides of the screen and a two-pixel paddle positioned at the bottom of the screen which the agent controls with the aim of catching the ball. When the falling pixel reaches the bottom of the screen the agent either gets a reward of 1 if the paddle overlaps with the ball and a reward of 0 otherwise”.

The result shows that, after 20 million frames of training, the agent can catch the “falling ball” roughly 85% of the time. The recurrent attention model was tested, and “since the agent was not in any way told to track the ball and was only rewarded for catching it, this result demonstrates the ability of the model to learn effective task-specific attention policies”.

The topic of the paper [11] might also be of interest, as the team proposes “a novel recurrent neural network model that is capable of extracting information from an image or video by adaptively selecting a sequence of regions or locations and only processing the selected regions at high resolution”. This can be a “cheaper” version of a machine vision algorithm using machine learning.

In Lippiello et al. It was discussed that both in a 2D and 3D system a high frame rate is very ideal for accuracy [12]. However, it also discusses that if high frame rate cannot be achieved then rather than a two-camera vision system a one camera vision system would reduce computational cost and with improvements and effort put into the prediction algorithm would function well. This was also mentioned in Buttazzo et al. [7]. This is similarly tested for a manipulator testbed for catching objects by Fernandez et al. [8].

Following on, Ribnick et al. discussed the localization of projectiles on its apparent motion in a stationary monocular view in 3D. It analyses sequential images and makes use of at minimum 3 images sticking to the least-squares algorithm [13]. Such a system is also used in Cigliano et al. where it uses a circle detection algorithm [14]. Ribnick et al proposed a Charge-Coupled Device camera mounted next to the manipulator whereas Cigliano et al. had one on top of a manipulator. Cigliano et al. also discussed how a stereo camera system is usually used for “RoboCup” competition humanoid robots.

The paper from Belousov [2] mentioned before, states that the catching heuristics theories are in fact optimal control policies. The method proposed relies on complex stochastic optimal theory rather than simple heuristics. Unfortunately, there is no “generally accepted model that explains empirical observations of human interception of airborne balls”.

Chapman’s theory LOT is introduced, as well as an enhanced version from McLeod [15] incorporating visual observations (generalised optic acceleration cancellation (GOAC)). McLeod adds a constraint to Chapman’s LOT, which is “to control the rate of horizontal rotation necessary to maintain fixation on the ball”.

The optimal control problem under uncertainty is implemented using:

- MPC approach
- CasADi to compute the derivatives and cost function
- Ipopt to carry out the non-linear optimisation

Their results show that when the agent can continuously track the ball, the heuristics approach holds; when the tracking is interrupted (agent turning away from the ball to run faster), the optimal control theory answers the problem.

The tracking interruption can also correspond to the object going out of the field of vue of a camera. This highlights the importance of control theory in our research.

Chapman’s strategy has been implemented in a simulation, adding the “catcher’s own acceleration and the visuo-motor delay” in [16]. The Chapman’s theory was proven to be “robust against the effect of the catcher’s own acceleration on the optical acceleration of the fly ball and the introduction of visuo-motor delays” and

“robust against air friction. Some minor observations, however, were in contradiction with the theory. A similar experiment has been conducted by Zaal and can be found in [17].

Kober et al. and Flash et al. mentioned that humans rely on motor primitives and reinforcement learning (RL) to learn new motor skills [18][19]. Kober et al. went on to show how reward-based RL can learn “single-stroke or rhythmic” tasks [20][21].

Ijspeert et al. suggested the use of nonlinear differential equations whereby two dynamic systems are used but one system drives the others for a more stable system [22]. This means the systems do not require to learn entire dynamical systems.

Previous work of Kim [23] used a iCub robot in the simulation environment and a 7-DOF robot arm (KUKA LWR 4+) in the real world experiment. Similarly, the work of Bäuml [24] implemented their proposed method by using a DLR-LWR- III arm with DLR-Hand- II which are a 7-DOF robot arm and a 12 DOF hand.

From the sensor’s aspect, works form Rollin’ Justin [25] [26] [27] used stereo cameras and IMU to determine the robot pose. They managed to achieve a catching rate of almost 80%. The work of Kim [23] which used iCub also had stereo cameras (double lens, bi-focal lens or 4 cameras - 2 per eye) according to the wiki of iCub [28].

Nemec et al. and Kober et al. compares the two approaches of using Dynamic motion principles and reinforcement learning.[18][29][20] In a machine vision system the use of Kalman filter OpenCV and ASUS Xtion PRO LIVE camera system are used for Disney research in a catch and juggling robot named Sky.[30] This is a multi-camera system. They also use OpenNI to track the human throwing the ball.

A more recent system by researchers at Princeton University, Google, Columbia University, and, Massachusetts Institute of Technology, authored by Zeng et al.[31] explores the tossing robot with the use of the UR5 manipulator as well as later on a catching system with another UR5 systems working in tandem. This research was published at the half-way point about one month (31 days) after we started our project and is a highly advanced system.

The components of recent works used can be summarized: simulation environment, robot arm, robot manipulator and visual sensors (cameras). The simulation environment can test algorithms in a virtual environment without real experiment setting up for the time efficient and cost. Different models can be created and transfer to the simulation environment. In the real application, a robot arm is need for the experiment. It will define the reaching space for the experiment. The manipulator contains the robot's catching ability and need to be selected according to the experiment specifications.

The fourth objective of the study (the first three objectives are detailed in section 0) is then to create a virtual environment for the robot to test the control approach and to make it learn machine learning policies.

3 Project management

3.1 Project initialisation

The project has been led by 3 novice robotics students for a period of 75 days.

During the project initialisation, a project breakdown structure composed of 6 main tasks has been considered. These main tasks are presented in section 3.2.

A thorough project risk assessment has also been carried out to prevent serious issues. It is displayed in Appendix B.

A weekly basis for the meetings with the supervisors has been chosen to let enough time for the students to work on their respective tasks, but also to monitor them and prevent them from going off-topic.

3.2 Task allocation

For each main task, a leader has been chosen in order to equally distribute the workload.

1. **Literature review – Chirantan**

This main task includes: literature review, study of influent papers with regard to ball catching research.

The Gantt is displayed in Figure C.2.

2. **Specifications – Yang**

This main task includes: analysis of the components, software, hardware used by the researchers (literature review) to meet the targets, analysis of the connection constraints (between components).

The Gantt is displayed in Figure C.3.

3. **Implementation strategy – Achille**

This main task includes: basic algorithms considerations, hardware and software testing (before implementation).

The Gantt is displayed in Figure C.4.

4. **Implementation and coding – Yang**

This main task includes: debugging, creation of ROS network, communication between software/hardware and other components, assembling.

It represents the main part of the project and is critical for the project.

The Gantt is displayed in Figure C.5.

5. **Testing verification validation – Achille**

This main task includes: tests to validate the behaviour of the systems, re-programming if needed.

The Gantt is displayed in Figure C.6.

6. **Deployment and documentation - Chirantan**

This main task includes: deployment of the system and write-up of the documentation.

The Gantt is displayed in Figure C.7.

Overall, the workload tended to the following distribution:

- Achille (40%)
- Yang (40%)
- Chirantan (20%)

3.3 Time management

With a time management technique, it has been planned to discuss with the supervisors and follow their guidelines in line with the deliverable dates for each progression of the project.

The comparison between the expected duration of the main tasks and their real duration is highlighted in Table 3.1.

Table 3.1 - Comparison main task expected duration and real duration

Task	Expected duration	Real duration
Literature review	1 week	10 days
Specifications	1 week	5 days
Implementation strategy	3 weeks	20 days
Implementation and coding	3 weeks	20 days
Testing verification validation	1 week	14 days
Deployment and documentation	1 week	6 days

The Milestones (mostly secondary reports) are described in each Gantt chart displayed in Figure C.1.

From Table 3.1, it can be noticed that the first three main tasks were completed in time. However, delay started to accumulate from the implementation strategy task due to compatibility and connection issues (which implied a non-negligible part of debug time). The implementation of the ROS control also added a significant delay. Finally, the reinforcement learning part has been slightly overlooked due to time constraints.

4 System specifications

4.1 Performance

The systems need to be able to process or compute data in order to predict the trajectory of a ball and probability of where and whether it will reach. Thus, the computer specs must be set and are necessary to train a reinforcement learning algorithm (or even a simple control algorithm).

4.2 External Interfaces

The inputs for the software system are the raw image data and measured distance which is captured by the stereo camera and ultrasonic sensor. The image data will give the information about current state containing the ball's existence, position in the image. That information will be used by the software system for estimating the position, velocity, and training the robot's optimal policy. The output is the action of the robot, which is its catching position. This position will be given to the motion planning function which can compute the inverse kinematics and then pass the results of required information about the joint angles to the controller to control the step motors.

4.3 Hardware System

The hardware system requires sensors, processors, computers, microcomputers, and vision sensors (passive and active).

4.3.1 Dobot Magician

The robot to be used is called "Dobot Magician". It is a 4 DOF desktop fit robotic manipulator arm. It has inbuilt software and hardware of its own. The forward and inverse kinematics of the robot are given in [32].

The specifications for the Dobot can be found in [33] and the dimensions are shown in Figure 4.1.

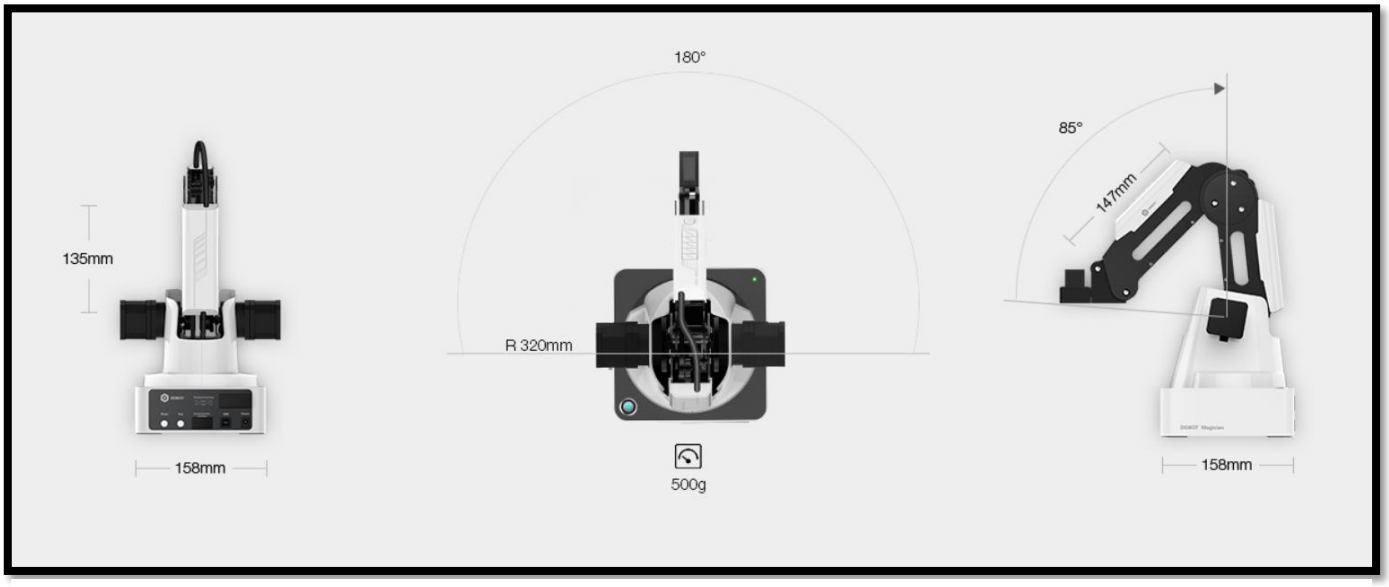


Figure 4.1 - Dobot Magician dimensions and joint limits

4.3.2 Microsoft Kinect

Microsoft Kinect version 2 has been used. It has a wide-angle time-of-flight camera (ToF camera) and can process 2 gigabits of data per second in order to read its environment, however only usb 3.0 is supported. According to its specification [34], the working range is defined in Figure 4.2. The working distance for depth detection is from 0.5m to 8m and the working angles are 0-70 degrees horizontally and 0-60 degrees vertically.

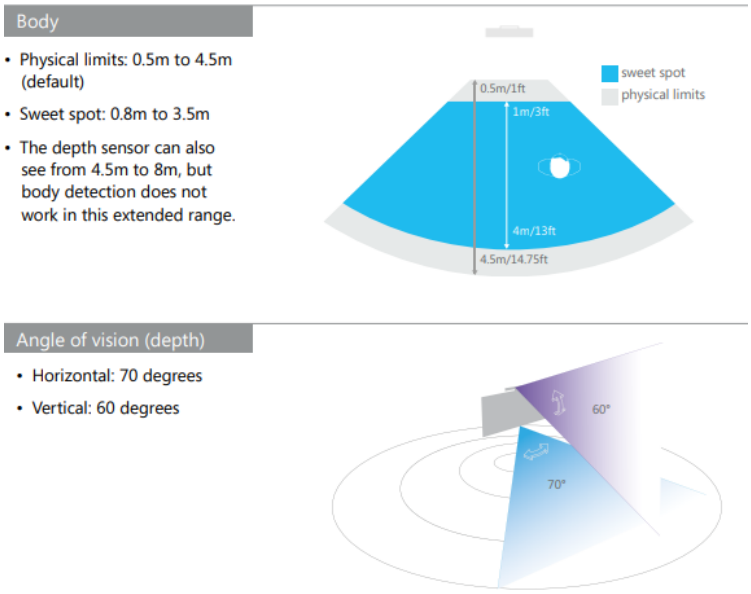


Figure 4.2 - Range of detection of Kinect V2

4.3.3 Raspberry Pi

Ubuntu OS (or other Linux OS) has been used on a computer to simulate and train machine learning/reinforcement learning algorithm.

A single-board computer (SBC) like “Raspberry Pi”, shown in Figure 4.3, has also been used to connect extra components such as sensors.

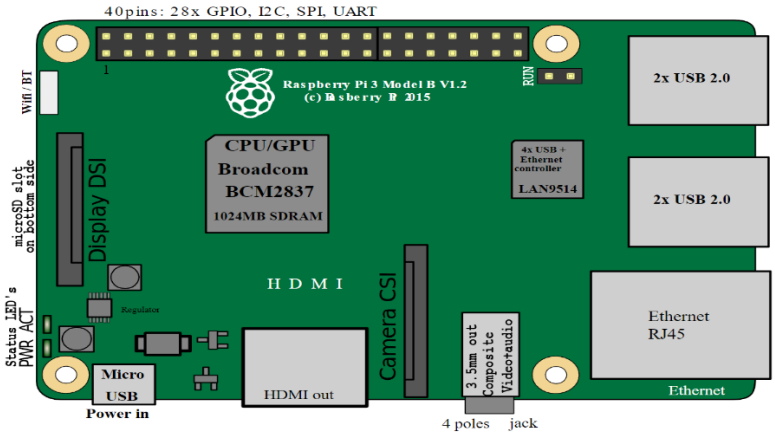


Figure 4.3 - A basic structure of a Raspberry Pi 3 Model B V1.2

The Raspberry Pi 3 Model B is the 1st model of the 3rd generation of Raspberry Pi. It replaced the Raspberry Pi 2 Model B. It was released in February 2016 [35].

It has the following specifications as indicated by the “Raspberry Pi Foundation” itself [35]:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data
- Upgraded switched Micro USB power source up to 2.5A

4.4 Functions

The “In-to-Net Bot” is defined by two main functions, based on Figure 4.4

The first main function is the ball catching function achieved by nonlinear optimisation control. The ball is detected by the sensors which give the necessary information to the controller which generated joint angles for the manipulator to catch the ball.

The second main function is the ball catching function achieved by Reinforcement Learning. The ball is still detected by the sensors which give the necessary information to the controller which tries joint angle configurations for the manipulator to catch the ball. Those configurations are updated by the new policy at each throw.

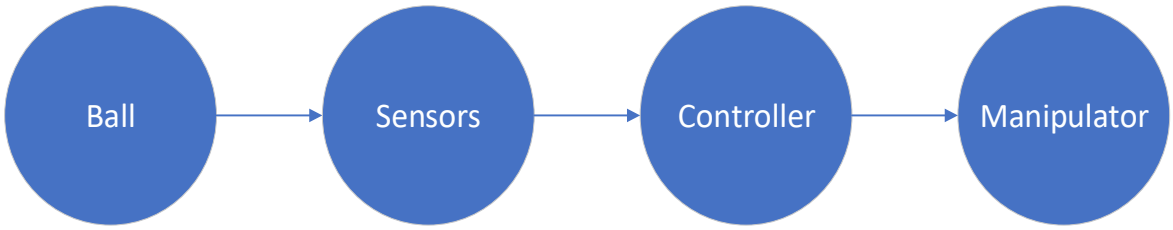


Figure 4.4 - Product main function basis

4.5 Architecture

ROS is used in the project. It is a robotics middleware which collects necessary software for robots. It provides services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management [36].

By researching those physical engines, Gazebo is selected as our training environment in the 3D space as Gazebo is already implemented in ROS [36] and there are full documentation explaining the techniques inside it.

The latest version of ROS is called Melodic Morenia, however the most widely used version is Kinetic Kame currently. The most common used and supported platform for installing ROS is Linux especially Ubuntu system [37]. The ROS Kinetic Kame is long term available for Ubuntu Xenial (16.04 LTS) and the recommended installation process is provided in [38].

In 2019, the work of X. Ling [39] used vision system to help robot harvesting tomato. Their system structures are divided into a hardware layer and software layer. Thus, similar approach can be made for our real time operating system, whose architecture is defined as Figure 4.5.

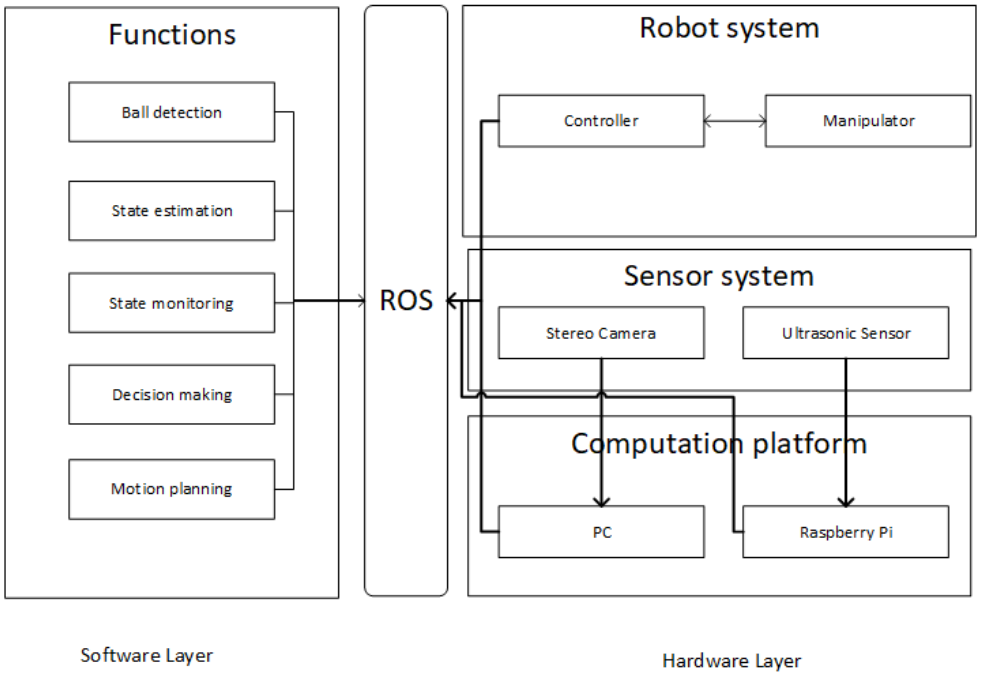


Figure 4.5 - Software architecture

The functions will receive the data from sensors and then process them on the PC or Raspberry Pi. ROS is the middleware connecting the hardware and software together. The functions relation are presented in Figure 4.6, the decision-making function is the one we need to train with machine learning or deep learning.

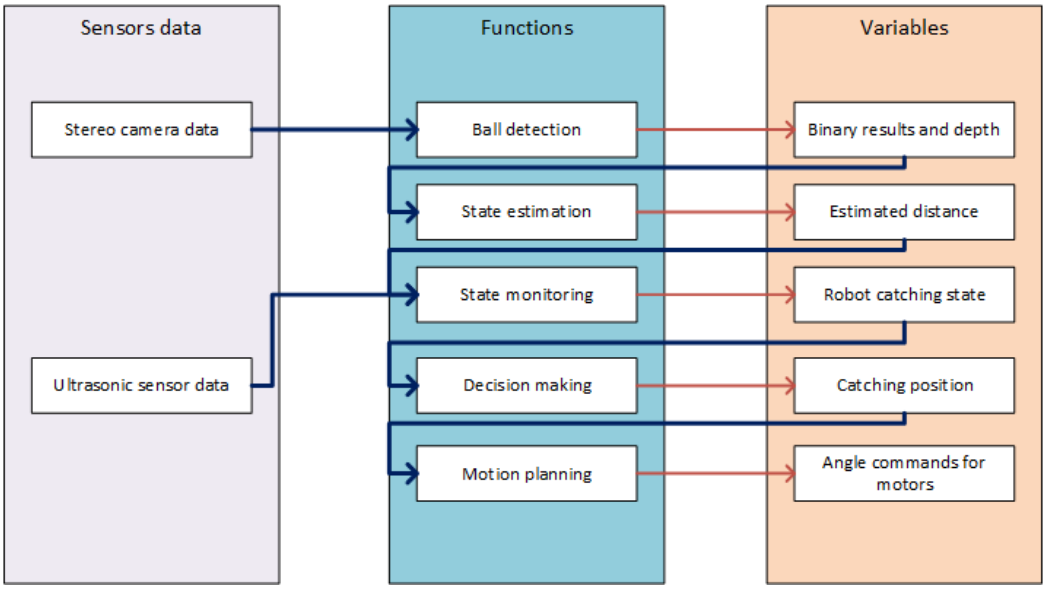


Figure 4.6 – Detail functions architecture

4.6 Implementation strategy

The system is divided into 3 main categories: The Robot system (Dobot Magician), the sensors and the computing platforms. They all interact with each other and with the external actors (the human and the ball). Figure 4.7 describes the interactions between each sub-system and the type of interfaces required.

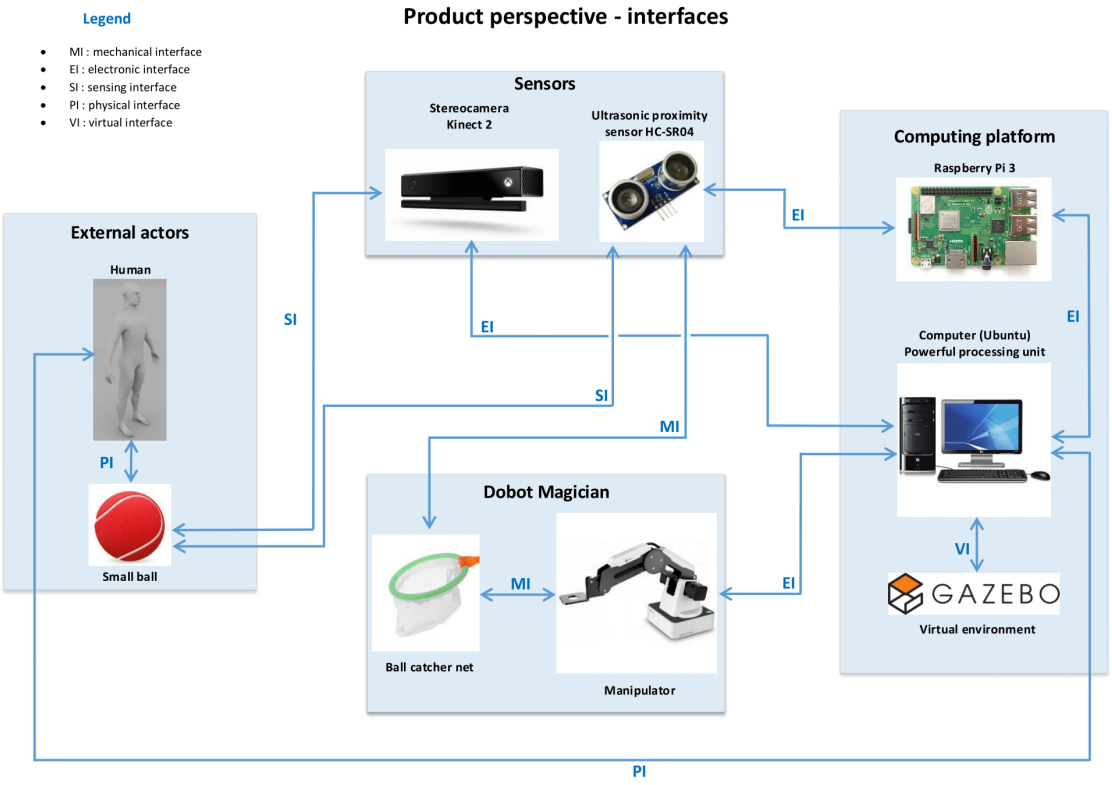


Figure 4.7 – Product perspective and interfaces

Table 4.1 describes the interfaces in detail.

Table 4.1 - Interfaces description

Interface	Type	Details
Human / Ball	PI	The Human or user grabs the ball to throw it at the Dobot
Human / Computer	PI	The Human can set up the robot using the computer. The user can also stop the experiment if needed.
Ball / Stereo camera	SI	The Kinect detects the ball and analyses features used for control / machine learning.
Stereo camera / Computer	EI	The Kinect is connected to the computer using a USB (3.0) cable and shares the data recorded about the ball (and the background).
Net / Manipulator	MI	The net is attached to the Manipulator and represents the end-effector the Dobot Magician. The ball thrown is caught in the net.
Manipulator / computer	EI	The Dobot is connect to the computer using a USB cable and is controlled using DobotStudio.
Net / Ultrasonic sensor	MI	The Ultrasonic sensor is fixed onto the net.
Ultrasonic sensor / Raspberry pi	EI	The Ultrasonic sensor is connected to the Raspberry PI to receive power and to share data to the computer. The connection is achieved using 1x 1kΩ resistor, 1x 2kΩ resistor, 1x power interface board, 8x jump wires. The set-up is explained in [40].
Ultrasonic sensor / Ball	SI	The Ultrasonic sensor detects the passage of the ball through the net.

Raspberry Pi / Computer	EI	The Raspberry Pi is connected to the computer through Wifi, but the communication needs to be set up using 1x Ethernet cable, 1x SD card, 1x Putty client for Windows (explanations in [41]).
Gazebo / Computer	VI	Gazebo is launched on the computer using ROS.

The main user interfaces displayed on the computer are:

- a custom DobotStudio interface to interact with the Dobot (and stop it in case an incident happens). It can also be a command prompt window.
- a custom window displaying the outputs of the Dobot training (for Machine Learning method) and the outputs of Dobot tests (for nonlinear control method).

The main user interfaces are aimed at the user, to help supervise the robot system.

Memory constraints are crucial for the Machine Learning method. The RL program cannot be used on the Raspberry Pi directly because it generates heavy computational cost. Moreover, Gazebo is used to train the RL algorithm in the virtual environment, however it requires Nvidia or ATI graphics cards [42]. Besides, camera information is needed to view the robot state through Rviz. Those requirements can't be met as Raspberry Pi doesn't have a powerful enough GPU or any USB 3.0 port. A computer meeting the requirements is therefore chosen to handle stereo-camera connection and Gazebo virtual environment display [42].

5 Implementation and testing

5.1 Overview of the system

The implementation has been divided into 2 main parts:

- **Physical ITNB** (composed of a Kinect version 2, a Dobot Magician with a net to catch the ball, a table tennis ball for testing – shown in Figure 5.1)

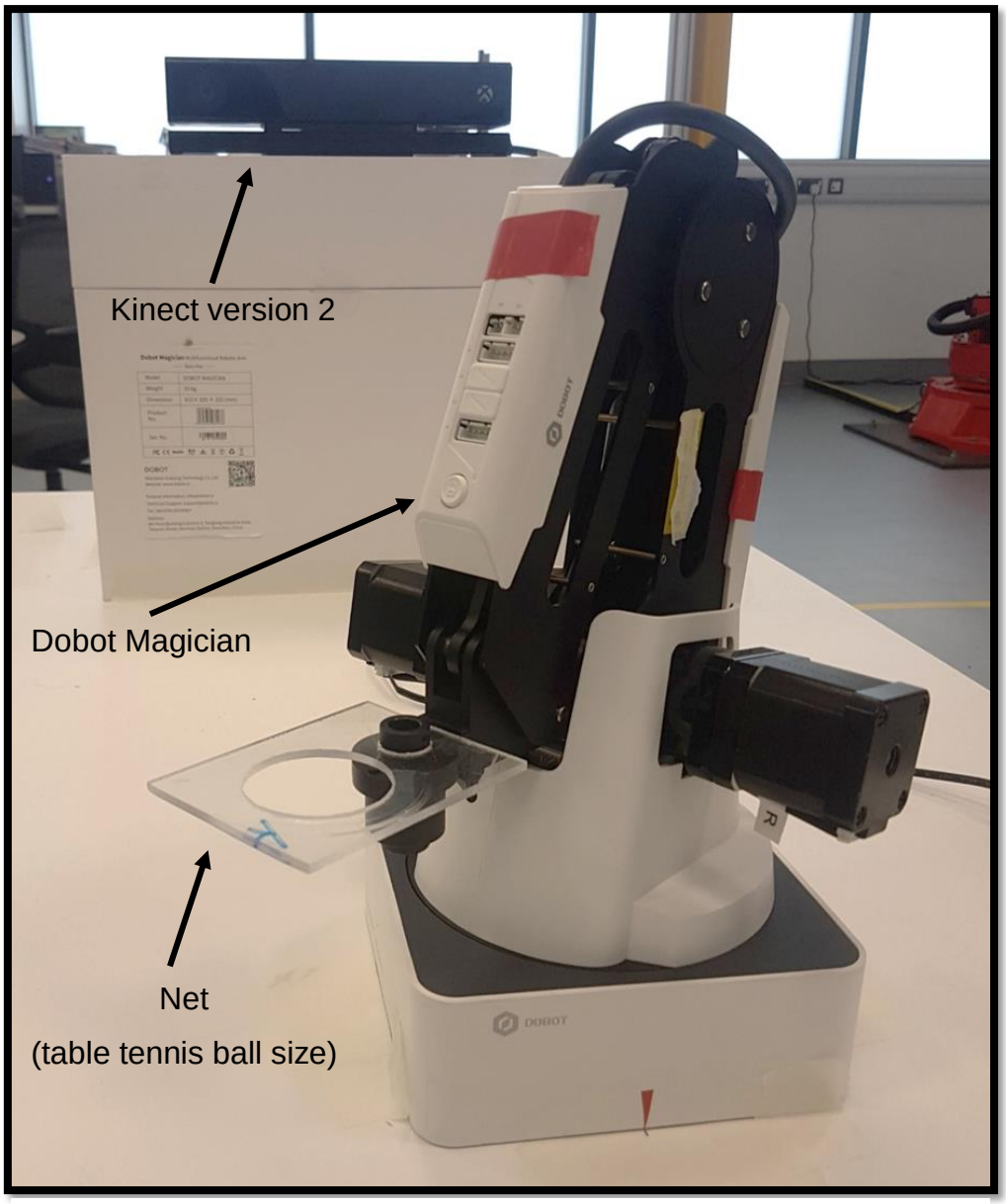


Figure 5.1 - Physical ITNB overview

- **Virtual ITNB** (composed of a mechanical model of the Dobot Magician with a net to catch the ball, a model of a table tennis ball, a Kinect Gazebo-ROS plugin – shown in Figure 5.2)

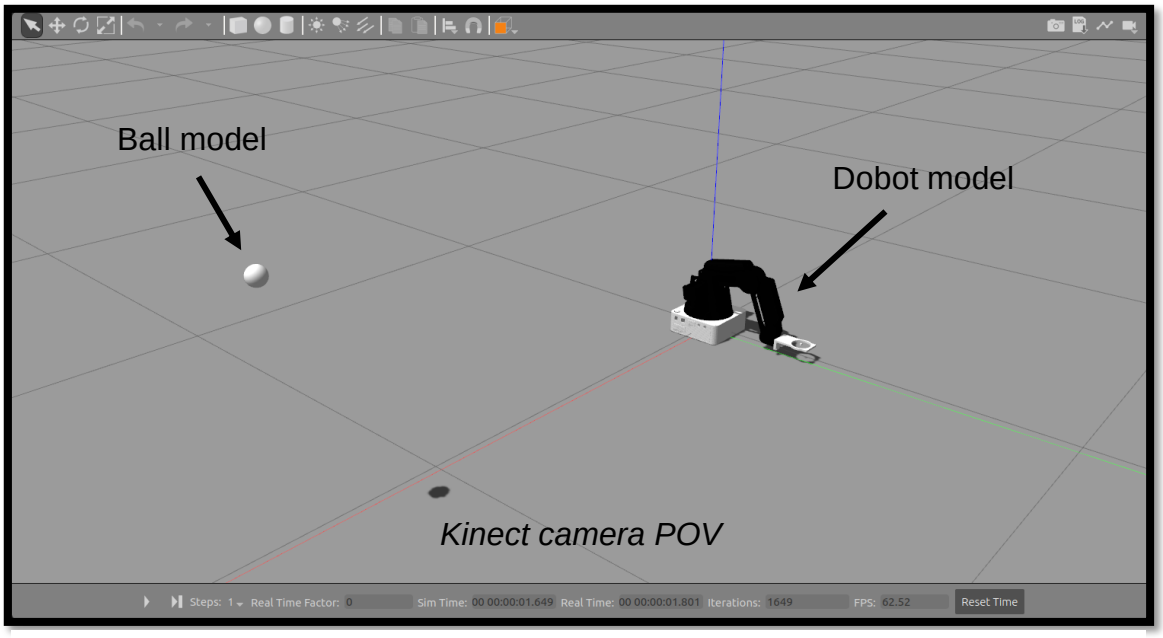


Figure 5.2 - Virtual ITNB overview

This step follows the implementation strategy step where the components and the software have been chosen and tested.

5.2 Physical ITNB

5.2.1 Mechanical structure

The ITNB main mechanical structure is based on the Dobot Magician structure whose specifications are shown in [43] and in Figure 4.1. It is composed of 4 degrees of freedom (4 DOF): one base joint, one shoulder joint, one elbow joint and one wrist joint to maintain the end-effector parallel to the “ground”, as shown in **Error! Reference source not found.**, extracted from [44].

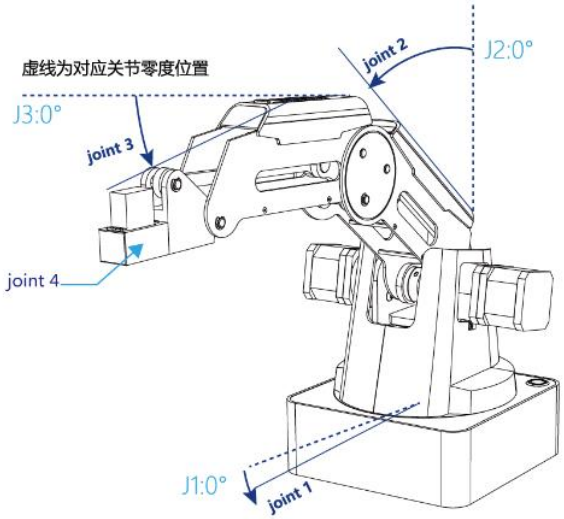


Figure 5.3 - DOBOT joint angles

The robot is made of ABS (Acrylonitrile Butadiene Styrene) engineering plastic and Aluminum Alloy 6061, therefore it is very robust. Moreover, its small size (height < 30cm) and its small weight (< 500 g) are adapted to laboratory testing without bulky security systems (fences, laser curtains…).

The net, shown in Figure 5.4, has been designed using Catia V5 to fit the Dobot end-effector attachment point (prismatic shape). The hole only has a 0.5mm extra diameter than a standard table tennis ball.

Due to 3D printing issues with the Dobot Magician (PLA material used), a temporary net made of Plexiglas and shown in Figure 5.1 has been used.

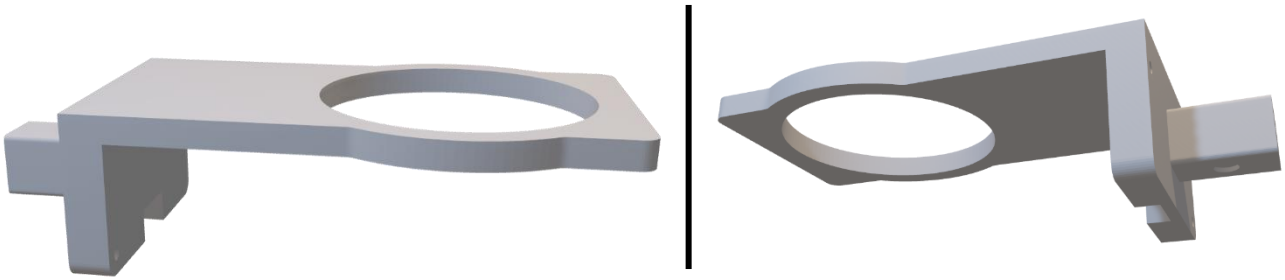


Figure 5.4 - ITNB Net CAD design

5.2.2 Joint control

As a first approach to control the position of the net, a forward kinematics model and an inverse kinematics model of the net have been considered.

The forward kinematics model ensures that the net is correctly moved in the 3D space, depending on joint angles. The modified Denavit-Hartenberg (DH) parameters [45] provide a parametrisation for the forward kinematics model. The modified DH parameters for the Dobot and net are shown in Figure 5.5.

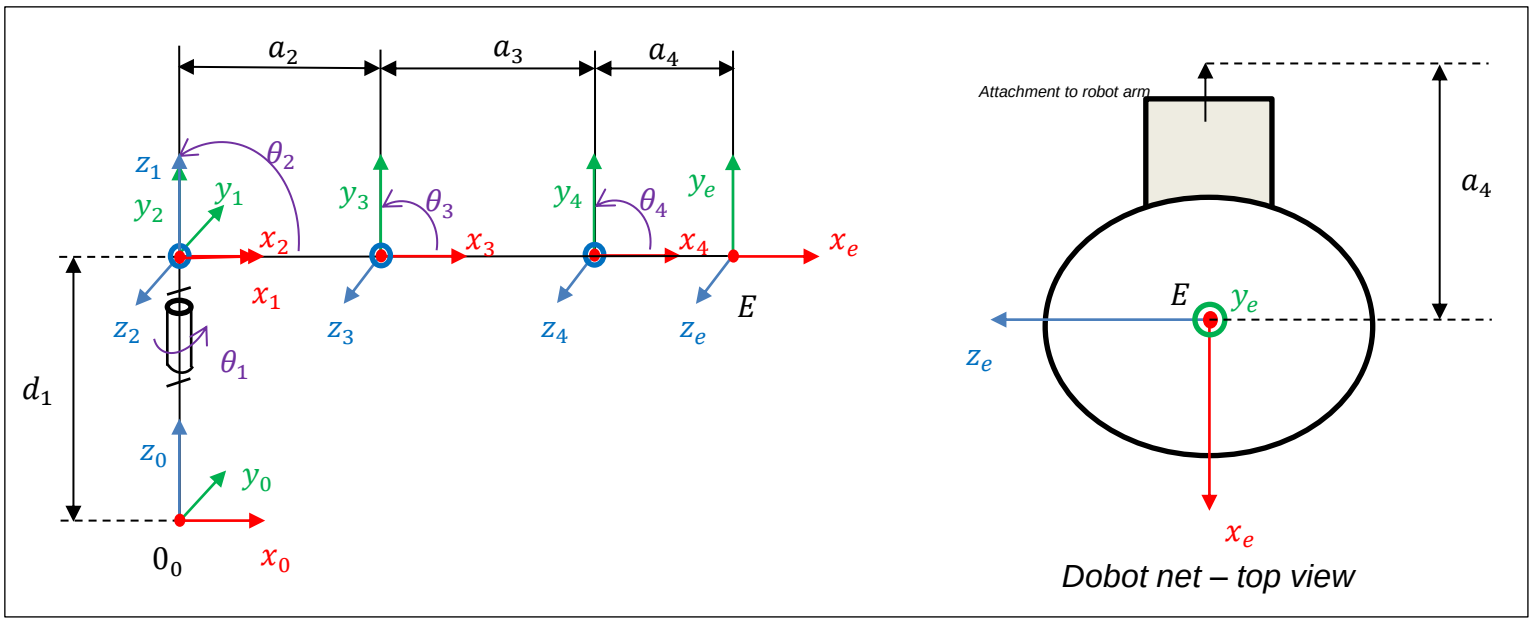


Figure 5.5 - Dobot forward kinematics model (DH parameters)

Table 5.1 gathers all the DH parameters needed to derive the forward kinematics.

Table 5.1 - DH parameters for Dobot Magician

Link i	α_{i-1} (degrees)	a_{i-1} (mm)	d_i (mm)	θ_i (degrees)
1	0	0	d_1	θ_1
2	α_1 = 90	0	0	θ_2
3	0	a_2	0	θ_3
4	0	a_3	0	θ_4
5	0	a_4	0	end-effector (net)

The dimensions of the Dobot Magician are indicated in Table 5.2.

Table 5.2 - DH parameter values

Parameter	Value
d_1	140 mm
a_2	135 mm
a_3	147 mm
a_4	Defined by the net (12 mm)

Using the transformation matrix from link {i-1} to link {i},

$$ {}^{i-1}_{i}T = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \sin(\theta_i)\cos(\alpha_{i-1}) & \cos(\theta_i)\cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -\sin(\alpha_{i-1})d_i \\ \sin(\theta_i)\sin(\alpha_{i-1}) & \cos(\theta_i)\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The transformation matrix from end-effector {5} to the base {0} can be obtained,

$$ {}^{0}_{5}T = {}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T\,{}^{3}_{4}T\,{}^{4}_{5}T $$

On the other hand, the objective of inverse kinematics is to determine the joint angles. The geometric method is used since it is the simplest method for this kind of problem. The method is inspired from [46].

A comprehensive explanation of the inverse kinematics equations is given in Appendix A.

As a second approach to connect the Dobot to the other components of the physical ITNB, a ROS network has been created. Specifically, a ROS node (explained in section 5.2.2) depending on Dobot ROS Service (supplied by Dobot customer service), has been created to control the robot arm. An end-effector offset has been included in the C++ functions to consider the position of the centre of the net (moved to a target position determined by the Kinect and its ROS node, explained in section 5.2.7).

The Dobot ROS API description (providing C++ functions) is downloadable and explained in [47].

5.2.3 Robot Perception Overview

The robot perception is a key part to achieve ball catching task in our project. The robot needs to have the object's position information in real-time to estimate ball's landing point.

The robot perception can be divided into two main components, object detection part and 3D coordinate transformation part respectively. The object detection part is used for detecting the object in real time while receiving images from Kinect 2 sensor. The second part is used for obtaining object's real-world coordination from the detection information.

5.2.4 Object detection - tracker

In 2015, a tracker with Kernelized Correlation Filters (KCF) was proposed by João F. Henriques [48] which showed promising results in object tracking. Thus, the first object detection attempt in our project is using an object tracker as mentioned above. Following the documentation of opencv tracker [49], the KCF tracker was implemented in our project. However, due to the unstable light condition and relative high computation demand of this filter, the tracking result's accuracy isn't sufficient for our high-speed flying ball application.

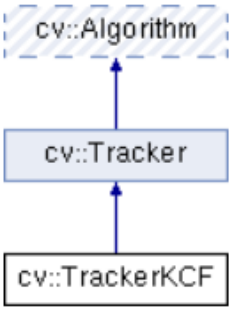


Figure 5.6 - OpenCV KCF tracker structure

5.2.5 Object detection – color detection in each frame

In order to have a more accurate detection of object, another solution of using color detection in each frame was implemented. In the previous work of Neves in 2015 [50], a framework of using RGB color and depth information to detect a flying ball has been raised. In our application, the object detection is also implemented by threshing the image's color. The color thresholding operations are done in HSV colorspace which is a model to represent the colorspace similar to RGB model [51].

The HSV stands for Hue, Staturation and Value in the colorspace, RGB stands for Red, Green and Green values in the colorspace as shown in Figure 5.7 and Figure 5.8.

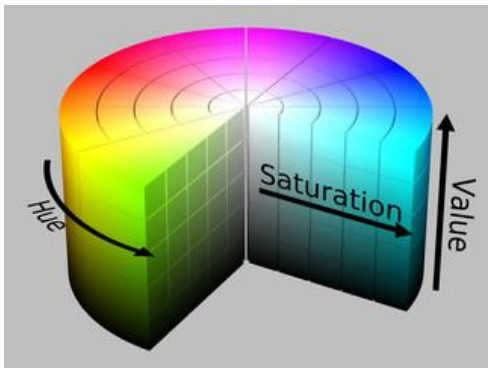


Figure 5.7 - HSV presentation

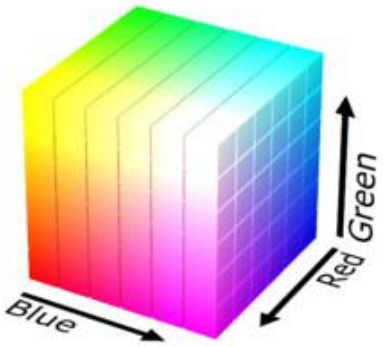


Figure 5.8 - RGB presentation

The first ball detection framework is illustrated in Figure 5.9. The detection is done in each frame by using the HoughCircle detection [52] on the thresholded image.

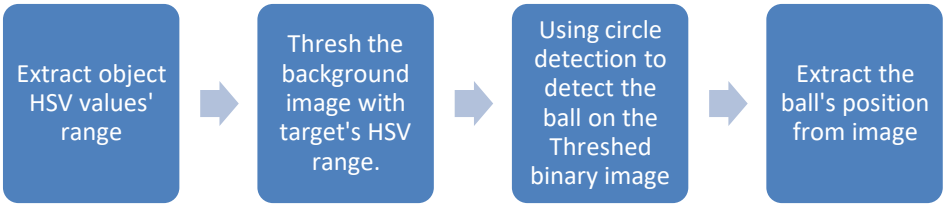


Figure 5.9 – First detection framework

Theoretically, the HoughCircle detection can perfectly detect the ball in each frame as it is a circle in the image. However, due to the light condition, the threshed object image could be a semi-circle or even not a circle. Due to that factor, a second detection framework which switched the detection mechanism to blob detection was purposed and implemented. The blob detection is basically extracting a group of pixels which has similar color or other features like circularity and convexity [53]. The second detection framework is illustrated as Figure 5.10.

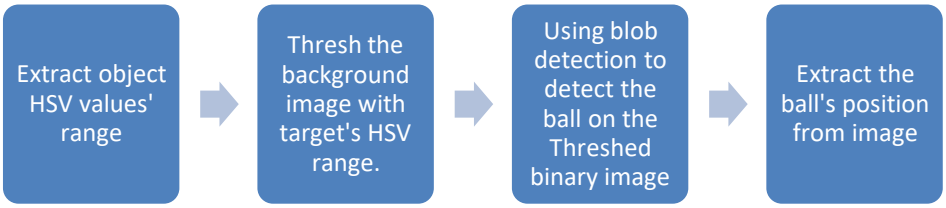


Figure 5.10 - Second detection framework

5.2.6 Depth acquisition and coordinates transformation

Although the detection part gives the position of ball on the image, it still isn't a real-world coordinate of the object. According to the description of Kinect 2 sensor working range and angles in Figure 4.2, two projections in x and y directions can be founded in Figure 5.11. The meanings of parameters are declared in Table 5.3.

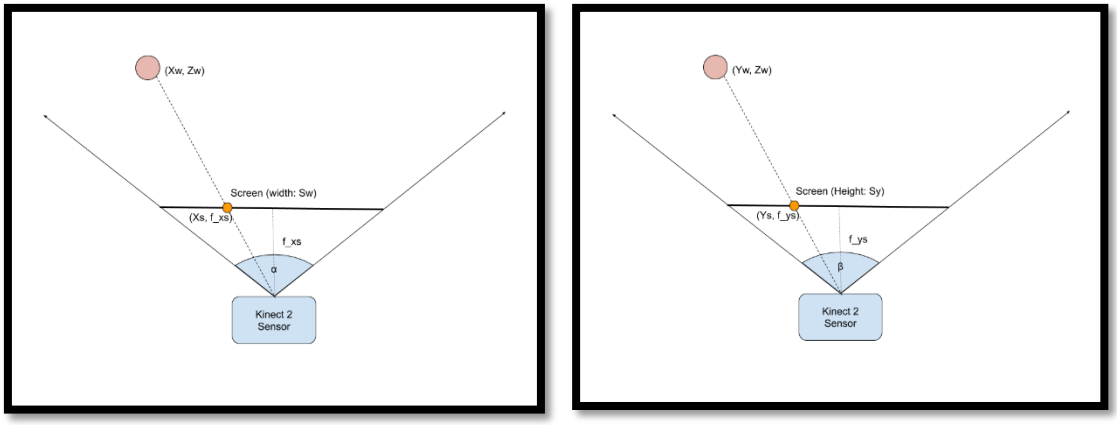


Figure 5.11 - Projections

Table 5.3 - Projections parameters

Parameter	Physical meaning	Values
α	Horizontal angle of vision	70 degrees
β	Vertical angle of vision	60 degrees
X_s	Object's screen position in x direction	
Y_s	Object's screen position in y direction	
S_w	Screen width	1920 pixels
S_y	Screen height	1080 pixels
F_{xs}	Focal length in width projection	
F_{ys}	Focal length in height projection	
X_w	Object's real-world position in x direction	
Y_w	Object's real-world position in y direction	
Z_w	Object's real-world position in z direction	

The focal lengths can be obtained by using screen's size and sensor's angle of vision information as follows.

$$\frac{\frac{1}{2} * S_w}{F_{xs}} = \tan\frac{\alpha}{2} \ , \qquad \frac{\frac{1}{2} * S_y}{F_{ys}} = \tan\frac{\beta}{2}$$

$$F_{xs} = \frac{\frac{1}{2} * S_w}{\tan\frac{\alpha}{2}}$$

$$F_{ys} = \frac{\frac{1}{2} * S_y}{\tan\frac{\beta}{2}}$$

During the detection, the object's screen position (X_s, Y_s) is obtained, the distance Z_w is also obtained by registration the depth information via librefreent2 driver [54]. Then, using similar triangles, the object's real-world position can be deduced as follows.

$$\frac{X_w}{X_s} = \frac{Z_w}{F_{xs}} \ , \qquad \frac{Y_w}{Y_s} = \frac{Z_w}{F_{ys}}$$

$$X_w = X_s * \frac{Z_w}{F_{xs}} \ , \qquad Y_w = Y_s * \frac{Z_w}{F_{ys}}$$

The final representation of X_s and Y_s can be obtained by replacing the deduced focal lengths, thus the expressions of object's real-world position are as presented below,

$$X_w = X_s * \frac{Z_w}{\frac{1}{2} * S_w} * \tan\frac{\alpha}{2}$$

$Y_w = Y_s * \frac{Z_w}{\frac{1}{2} * S_y} * \tan\frac{\beta}{2}$Figure 4.2 - Range of detection of Kinect V2

Z_wFigure 4.2 - Range of detection of Kinect V2Figure 4.2 - Range of detection of Kinect V2

5.2.7 Ball trajectory estimation

The ball trajectory estimation task has been performed using Kinect version 2 to retrieve 2 frames from a live video feed, then extract the initial position and initial velocity vector to feed 3D trajectory models. The 3D trajectory models can be obtained from 2D trajectory models by rotating the 2D plane around the axis perpendicular to the "ground" (for instance in Figure 5.12, the rotation axis could be the $\vec{y}$ axis).

Two different approaches have been considered for the ball trajectory:

- **Ball motion without air resistance**

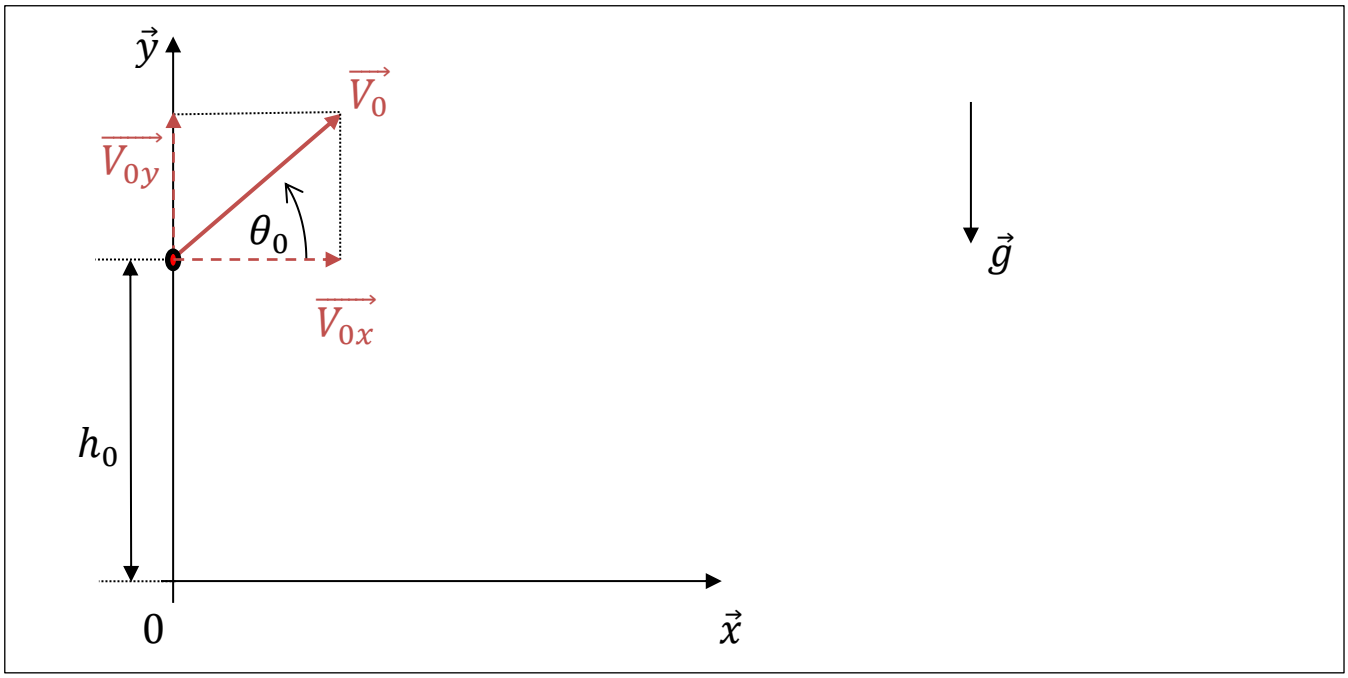


Figure 5.12 - 2D initial conditions for ball trajectory

According to Newton's 2nd law of motion,

$$\sum \vec{F} = m\vec{a}_G$$

The projection onto the axes $\vec{x}$ and $\vec{y}$ gives,

$$ma_x = 0 \Leftrightarrow \boldsymbol{a_x = 0}$$
$$ma_y = -mg \Leftrightarrow \boldsymbol{a_y = -g}$$

Integrating the acceleration,

$$\boldsymbol{v_x} = a_x t + constant = constant = v_x(t=0) = v_{0x} = \boldsymbol{v_0 \cos(\theta_0)}$$
$$\boldsymbol{v_y} = a_y t + constant = -gt + constant = \boldsymbol{-gt + v_0 \sin(\theta_0)} \quad [using: v_y(t=0) = v_{0y} = v_0 \sin(\theta_0)]$$

Integrating the velocity,

$$\boldsymbol{x} = v_x t + constant = \boldsymbol{v_0 \cos(\theta_0)\, t} \quad [using: x(t=0) = 0]$$
$$\boldsymbol{y} = -\frac{1}{2} g t^2 + v_y t + constant = \boldsymbol{-\frac{1}{2} g t^2 + v_0 \sin(\theta_0)\, t + h_0} \quad [using: y(t=0) = h_0]$$

The coordinates ($\boldsymbol{x}$, $\boldsymbol{y}$) represent the ball position at time t in Figure 5.12.

- **Ball motion with air resistance**

In order to determine the representation of the air resistance, the Reynolds number R_e must be calculated (cf. [55]).

$$R_e = \frac{\rho l v}{\eta}$$

Where ρ is the fluid density (in the study case, the air density), l is the characteristic cross-sectional length (the length of the fluid flow, in the study case it is the diameter of the ball), v is the velocity relative to the fluid (in the study case it can be approximated by the initial velocity of the ball v_0), η is the fluid viscosity (in the study case, the viscosity of the air).

The representation of air resistance depends on R_e:

- If R_e << 1, the air resistance is supposed to be linear
- If R_e > 1000, the air resistance is supposed to be quadratic
- In between, in depends on the case.

The force of air resistance (linear case) can be represented by,

$$\overrightarrow{F_{resist}} = -c\vec{v}$$

The coefficient c is the air resistance coefficient,

$$\overrightarrow{F_{resist}} = -c\vec{v} = -\frac{1}{2}\rho c_D A\vec{v}$$

Where c_D is the drag coefficient [56] and can be calculated using Brown and Lawler formula for $R_e < 2 \times 10^5$ (considered to be always verified in the study case),

$$c_D = \frac{24}{R_e}\left(1 + 0.15{R_e}^{0.681}\right) + \frac{0.407}{1 + 8710{R_e}^{-1}}$$

A is the cross-sectional area facing the flow (in the study case it is the surface of the circle whose diameter is the same as the one of the ball), ρ is the fluid density.

From Newton's second law of motion, adding the force of air resistance,

$$ma_x = -cv_x \Leftrightarrow \frac{\boldsymbol{dv_x}}{\boldsymbol{dt}} = -\frac{\boldsymbol{c}}{\boldsymbol{m}}\boldsymbol{v_x}$$
$$ma_y = -mg - cv_y \Leftrightarrow \frac{\boldsymbol{dv_y}}{\boldsymbol{dt}} = -\boldsymbol{g} - \frac{\boldsymbol{c}}{\boldsymbol{m}}\boldsymbol{v_y}$$

This system of differential equations can be solved using "odeint" [57] in python language for instance.

The force of air resistance (quadratic case) can be represented by,

$$\overrightarrow{F_{resist}} = -c\|\vec{v}\|\vec{v}$$

The equations can be solved as shown in [58], but the python approach using "odeint" is chosen because faster.

From Newton's second law of motion,

$$ma_x = -c\left(\sqrt{{v_x}^2 + {v_y}^2}\right)v_x \Leftrightarrow \boldsymbol{\frac{dv_x}{dt} = -\frac{c}{m}\left(\sqrt{{v_x}^2 + {v_y}^2}\right)v_x}$$

$$ma_y = -mg - c\left(\sqrt{{v_x}^2 + {v_y}^2}\right)v_y \Leftrightarrow \boldsymbol{\frac{dv_y}{dt} = -g - \frac{c}{m}\left(\sqrt{{v_x}^2 + {v_y}^2}\right)v_y}$$

Both approaches have been gathered into one realistic test, shown in Figure 5.13. The initial conditions for the simulation of the ball trajectory are the following,

- $d = 4.0 \times 10^{-2}\, m$ (diameter of the "table tennis" ball)
- $v_0 = 2.5\, m/s$
- $\theta_0 = 1.2\, rad$
- $h_0 = 1.2\, m$
- $g = 9.81\, m/s^2$

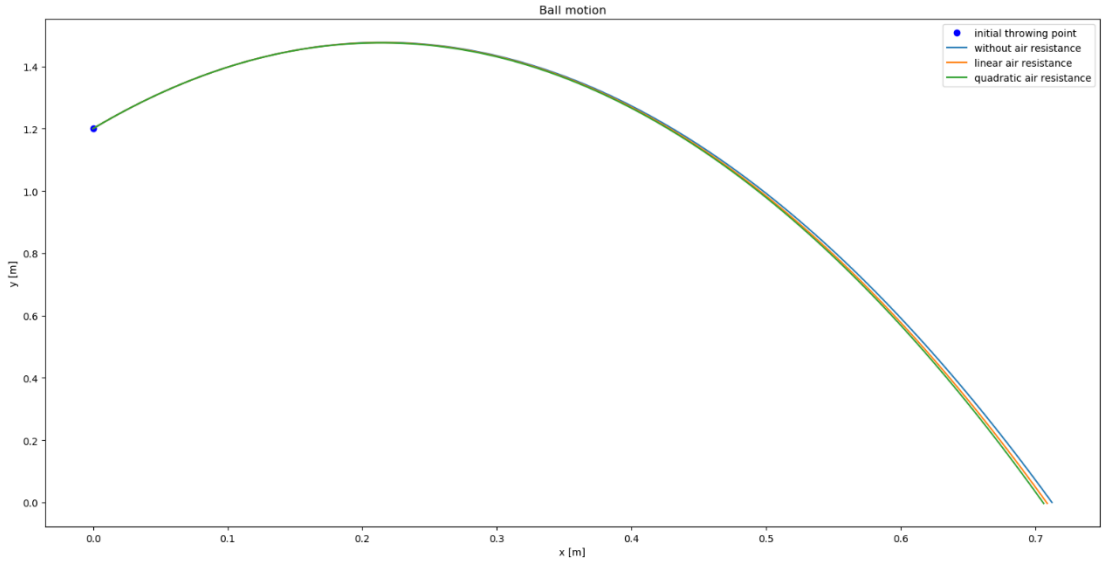


Figure 5.13 - simulation of ball trajectory

According to the similar results for a throwing distance < 2m over multiple tests, the chosen approach has been the trajectory model without air resistance, in order to speed up the calculation and therefore the communication between the components.

5.2.8 ROS communication

- RQT graph
- C++ / python node connexion

ROS contains multiple components including Nodes, Master, Parameter server, Messages, Topics, Services and Bags [59]. A ROS computational graph is presented in Figure 5.14. Nodes are the basic function unit for ROS and support C++ and python languages. Master is a central process that enable different nodes can find each other in order to communicate, the communication is done by sending and receiving data to or from a topic.

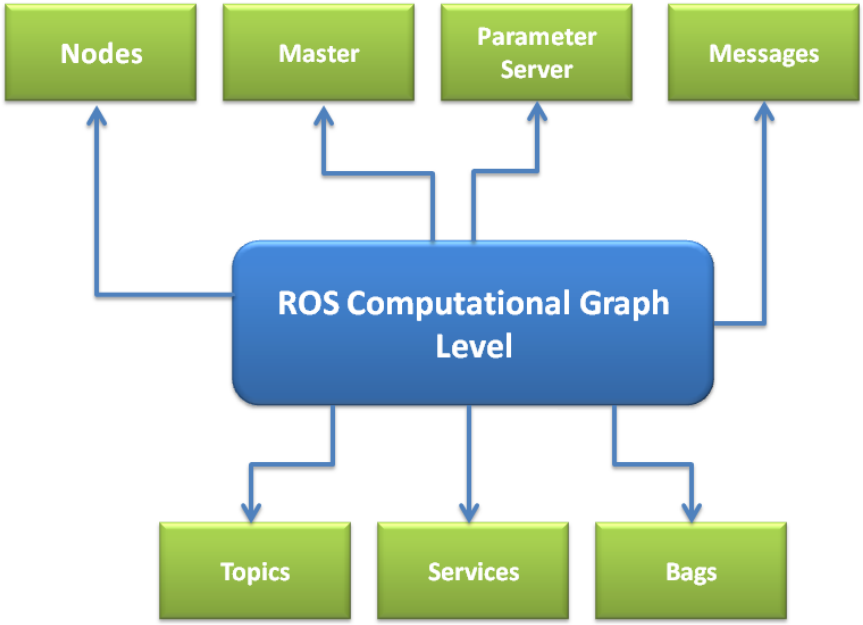


Figure 5.14 - ROS computational Graph Level

In our application, the communication is realized by two types of nodes, they are publishing node and subscribing node specifically. The publishing nodes will publish data to object position and estimation position topics, then the subscribing nodes will subscribe to those topics and use those data to control the robot. Our ROS communication structure is presented in Figure 5.15.

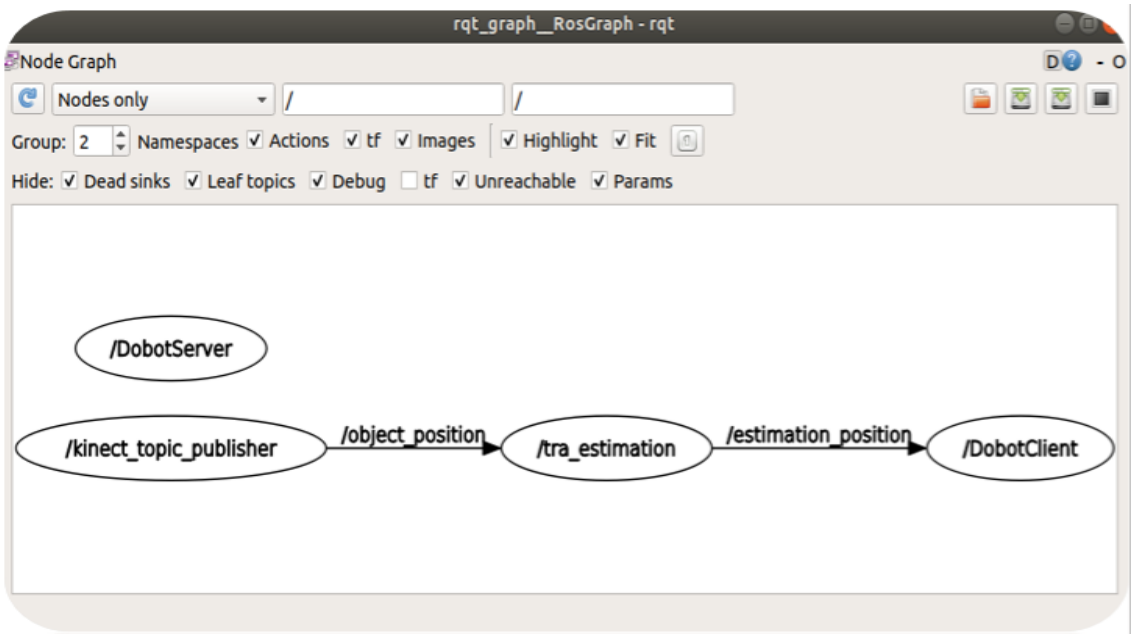


Figure 5.15 - ROS communication structure

For the robot control part, there are interactions of request and reply existing and those interactions can be implemented via ROS Service as DobotServer in Figure 5.15.

5.2.9 Reinforcement learning perspective

An ultrasonic sensor (Hc-Sr04 – presented in Figure 5.16) has been added to help the robot know when the ball successfully went through the net during reinforcement learning process. The sensor has been connected to a raspberry PI to integrate it to the ROS network created.

The connection process is explained in [40].

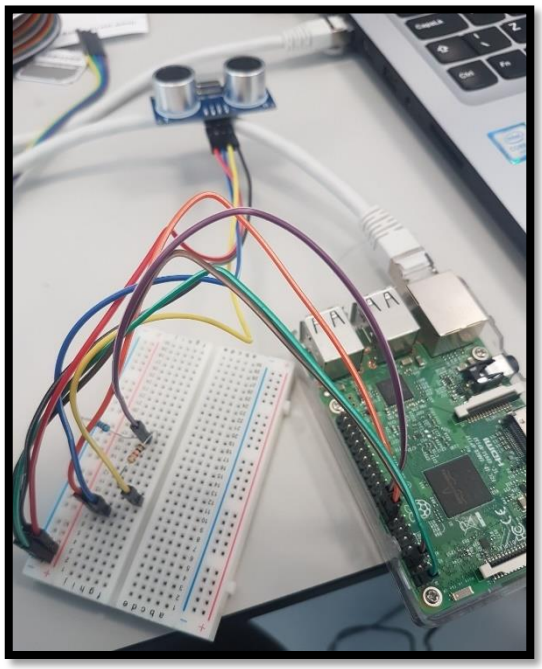


Figure 5.16 - Hc-Sr04 sensor and Ultrasonic sensor connected to Rapsberry pi

The sensor can be easily screwed under the net as shown in Figure 5.17 (to avoid detection if the ball bounces on the ring of the net).

According to the tests performed using signal analysis, the range of the sensor is: 2.5cm to 380cm. Thus, the hole on the net has been offset from the Dobot attachment point to ensure that the ball will not fall in the “dead zone” of the sensor.

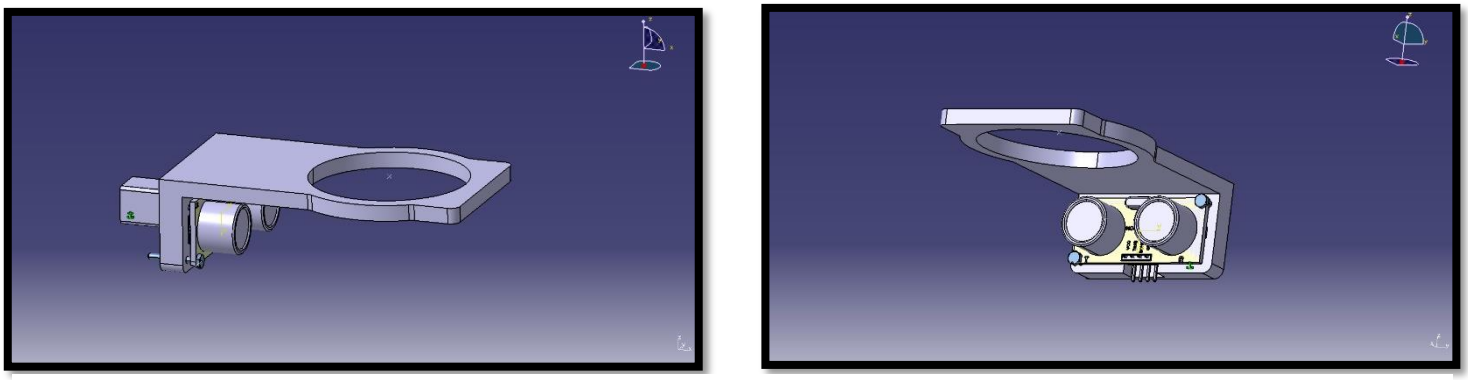


Figure 5.17 - ITNB net and ultrasonic sensor

The Raspberry Pi and HC-SR04 sonar sensor are operational for reinforcement learning applications using the physical ITNB. However, the net shown in Figure 5.17 has not been produced yet (a replacement has been in use for experimental purposes, as shown in Figure 5.1).

5.3 Virtual ITNB

5.3.1 Mechanical model in Gazebo

The CAD model of the Dobot Magician has been supplied by the Dobot customer support. The net, described in Figure 5.4, has been added to the robot arm using Catia V5. The assembly is shown in Figure 5.18.

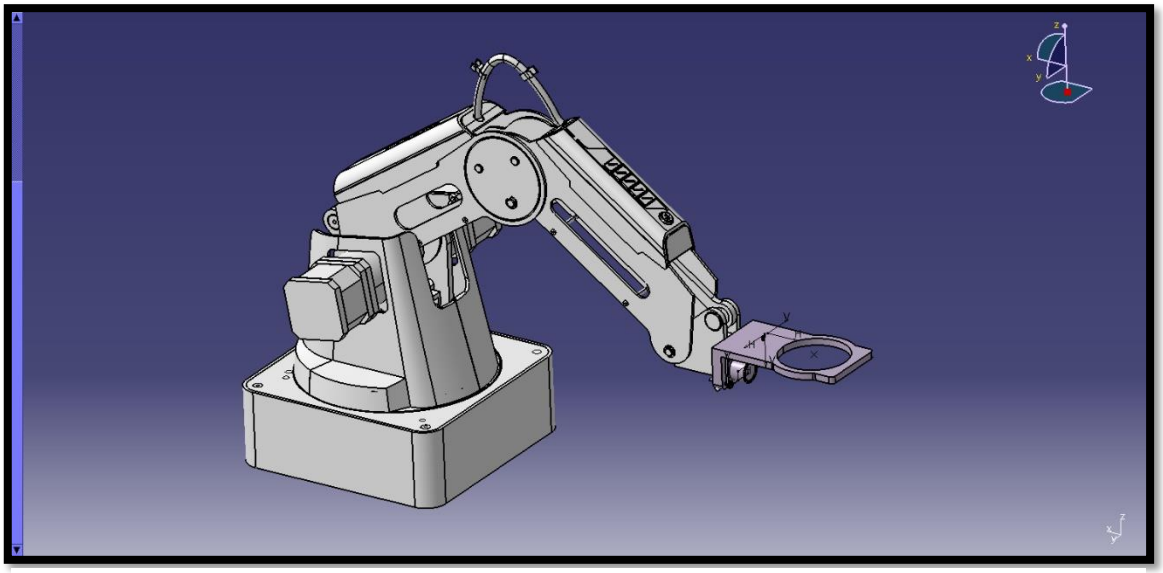


Figure 5.18 - Dobot Magician and net assembled

However, the CAD materials and the inertial matrices have not been provided by the Dobot customer support. The materials (described in section 5.2.1) have been created using Catia V5 and the inertial matrices have been calculated using MeshLab [60] and Blender [61].

The results from MeshLab were obtained considering homogeneous mass repartition and were cross-checked with the results from Catia V5 (where materials were applied).

Blender has been used to create a URDF description [62] of the ITNB. This description is supposed to be universal but can only be applied to open-kinematics chains. The ITNB has been considered as an open-kinematics chain even if the Dobot is composed of a closed-kinematics chain (using links to

maintain the end-effector parallel to the “ground”). The robot can then be spawned in Gazebo [62] using the URDF description.

5.3.2 Joint control in with Rviz and Gazebo-ros control

Once the robot has been spawned in the Gazebo environment, the joints are not controlled by default.

A simple solution is to add a gazebo-ros controller [63]. This controller can take the form of a PID controller. It is manageable for robots with a few DOFs. For instance, a sliding net can be controller using a PID controller tuned with rqt (software framework of ROS that implements the various GUI tools in the form of plugins [64]). The sliding net along $\vec{x}$ axis, shown in Figure 5.19, has been implemented to run the first reinforcement learning tests. The sliding net along $\vec{x}$ axis has been implemented to run the first reinforcement learning tests (1D movement of the net and 2D movement of the ball).

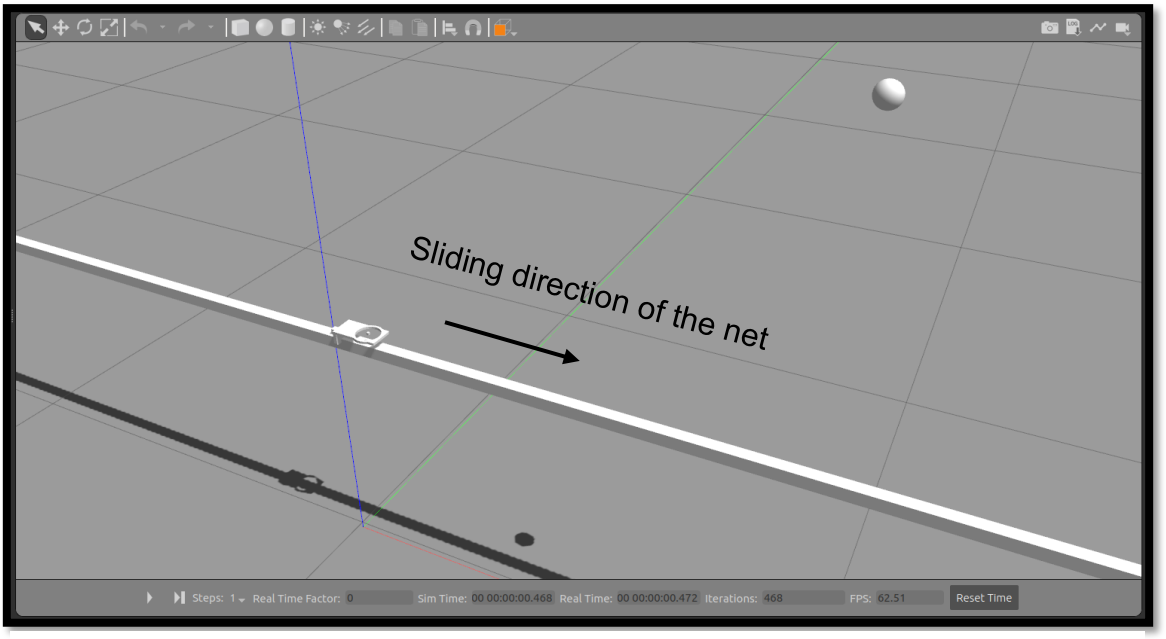


Figure 5.19 - Sliding net in Gazebo

The simple control concept in Gazebo can be extended to open-kinematics manipulators using Rviz [65] and Moveit! [66], a motion planning framework for ROS. The control of closed-kinematics manipulators is not yet handled by those

frameworks. Consequently, the ITNB has been modelled with 4 joints (last joint for the end-effector supposed to be maintained parallel to the “ground”). The result of the joint control is shown in Figure 5.20. The red cube represents the centre of the net and is moved to a desired target position (determined by the ball trajectory estimation).

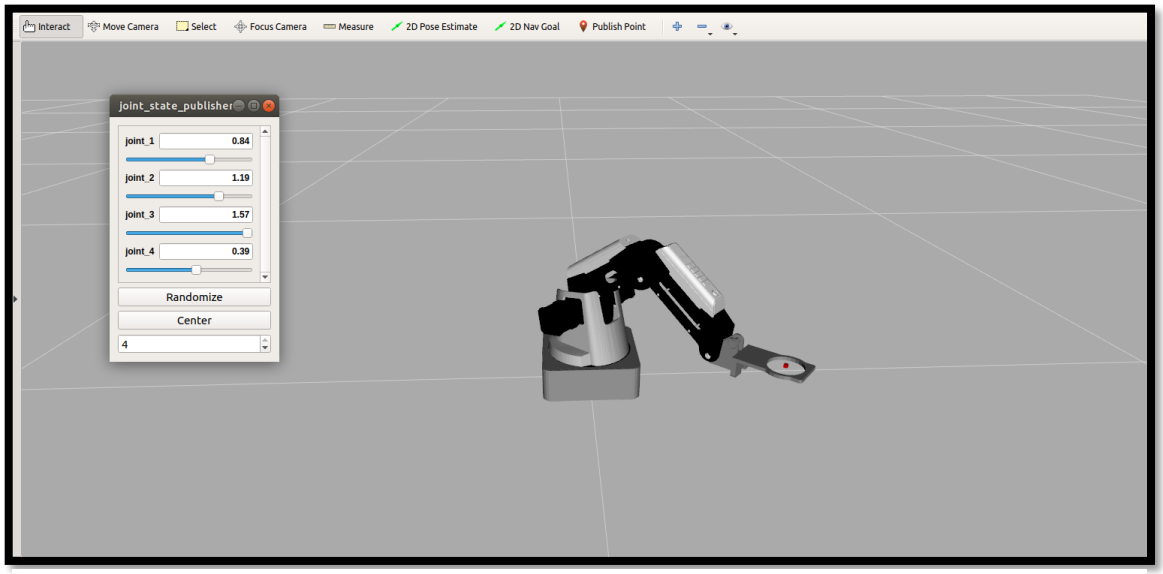


Figure 5.20 - Rviz model of virtual ITNB

Rviz and Gazebo can exchange data through ROS nodes: Gazebo tells Rviz the target position of the end-effector, and Rviz sends back the joint angles to reach the desired target.

5.3.3 Ball motion in Gazebo

Two approaches have been considered to describe the ball motion in Gazebo:

- **Model creation and C++ programming using ROS nodes**

This approach is complex because it involves the creation of multiple ROS nodes interfaced with Gazebo to describe the behaviour of the ball and to spawn new models of the ball once a ball has been thrown.

It is however a comprehensive way of coding the ball motion since it includes collisions and uses the physics engine within Gazebo.

- **Animation definition using Gazebo node**

On the other hand, a very simple method to describe the movement of a ball is by using the animation tool [67] provided by Gazebo (using a Gazebo node).

The main drawback is that the physics engine is not solicited by the animation, which means that the ball will not bounce. Still, the ball can be seen by a ROS-kinect plugin, which makes this approach interesting.

The ball motion (animation) is shown in Figure 5.19, along with the sliding net.

5.3.4 Reinforcement Learning in 2D environment and Gazebo

A 2D prototype environment was made to simulate a ball catching robot as shown in Figure 5.21. This environment is created by using pygame [68] which is an open Source python programming language library for making multimedia applications like games in 2D.

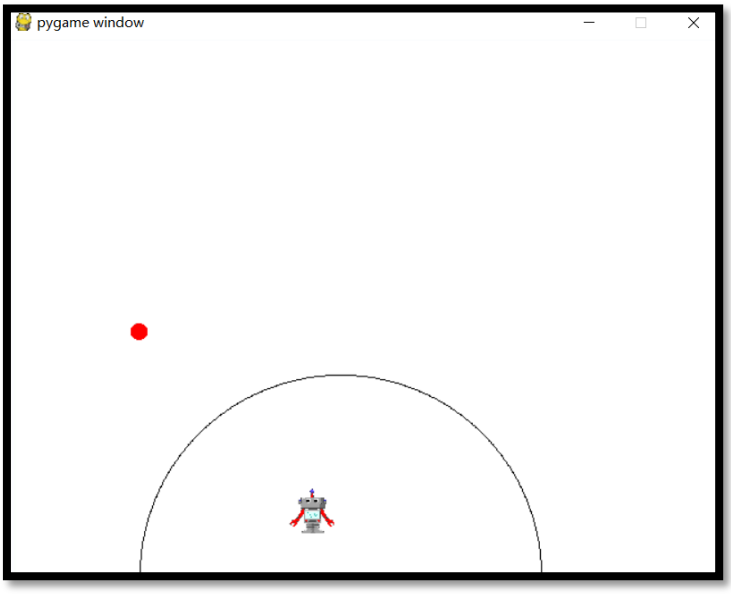


Figure 5.21 - 2D pygame environment

In the 2D environment, a red ball will drop from the left side with a random velocity, its target position is in the range of semi-circle. The agent (robot) can only move

inside the semi-circle range. The agent's task is trying to catch the ball before it landed inside the moving range

The implemented algorithm is based on ACKTR which is proposed by Yuhuai Wu in 2017 [69]. This algorithm is an improvement of actor-critic [70] whose structure is presented in Figure 5.22. One improvement is adding multiple workers which can be understood as sub-environments instead of just having one environment, that permits agents can explore the same environments asynchronously which can be seen in Figure 5.23.

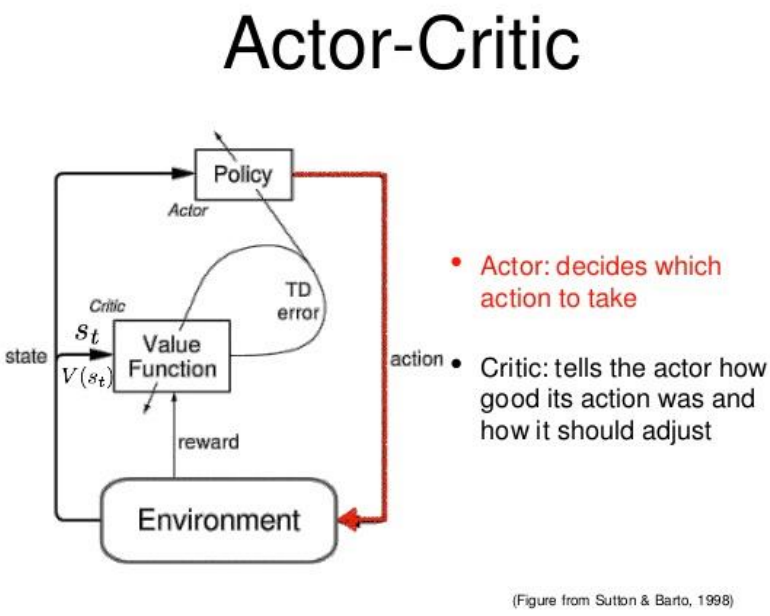


Figure 5.22 - Actor-Critic structure

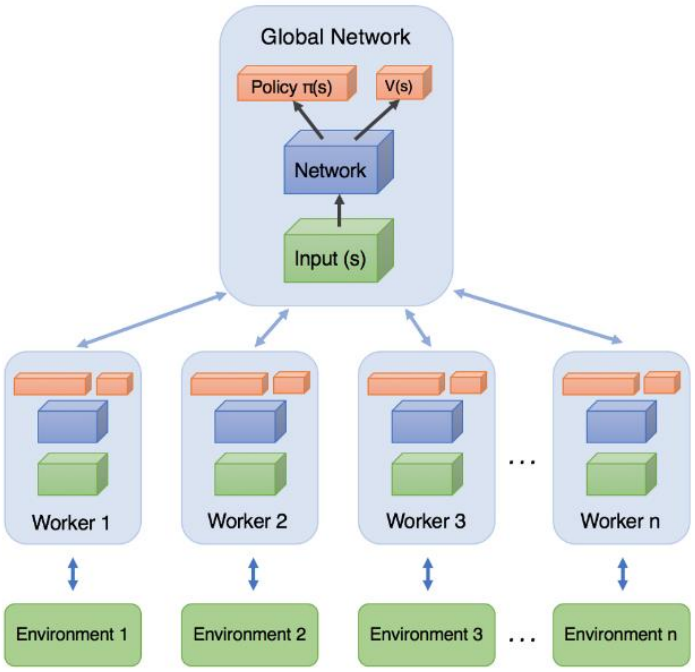


Figure 5.23 - Asynchronous Actor Critic

A gym-gazebo toolkit has been used for the gazebo environment and enables the use of gym which is an Openai reinforcement learning toolkit.

6 Validation

6.1 Physical ITNB

6.1.1 Robot control results

In order to validate the behaviours of real robot control, a 10 points data set are tested repeatedly to exam the repeatability of dobot, the tested data can be found in the GitLab repository/Blackboard repository. To visualize those testing data, the target points and reached points are plotted and presented in Figure 6.1, according to the confidence interval, a 95% confident error interval of $5.1787 \pm 1.3081\ mm$ is deduced by using similar approach of positioning error analysis of Chen.J's work in 1986 [71].

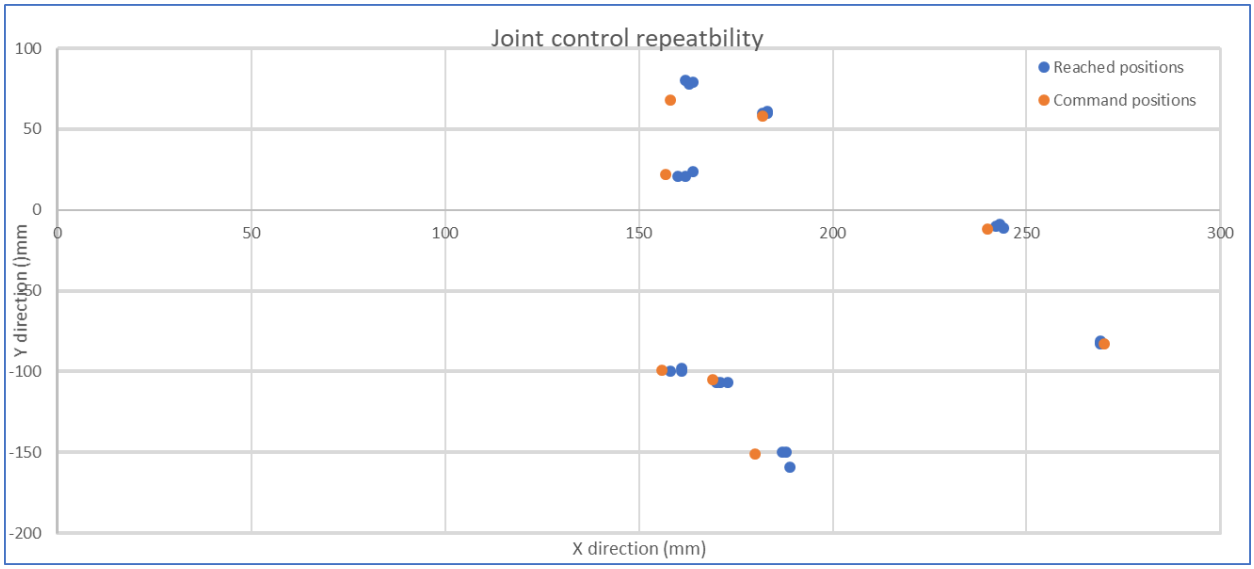


Figure 6.1 - Joint control repeatability

Although small errors can be found from the visualization and error interval results, the Dobot claims that the error could be limited in 0.2mm [72], which is much smaller than our result. There are several potential reasons causing that result, such as the links are rigid bodies, the servo motor angular position is influenced by speed (high speed overshoots angle), the precision of tools (we used the type measurement whose precision is 1mm) and the alignment of the pen and pen holding for drawing the points. Thus, a more precise measurement

such as laser measurement can be considered as an improvement method in the future, angle measurement could also be tested.

6.1.2 Object detection results

As explained in section 5, two detection methods which are static condition (static ball) and dynamic condition (flying ball) are implemented in our application. The circle detection and blob detection methods are tested in static and dynamic condition separately to exam their performance. Their performances are valued by frame per second (fps), success rate and error interval.

In both static and dynamic conditions, the blob detection method outperformed circle detection method in processed frames per second as shown in Figure 6.2 and Figure 6.3.

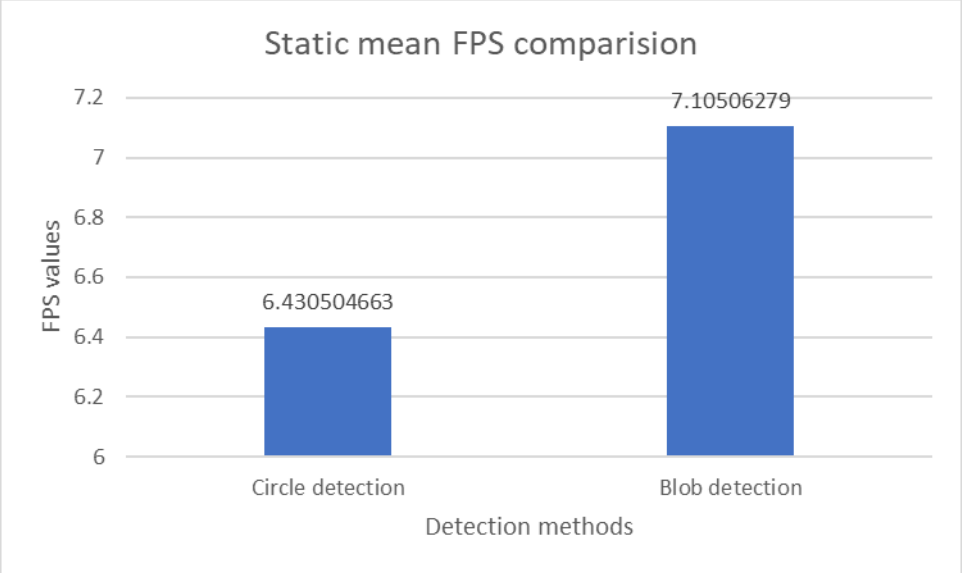


Figure 6.2 - Static mean fps comparison

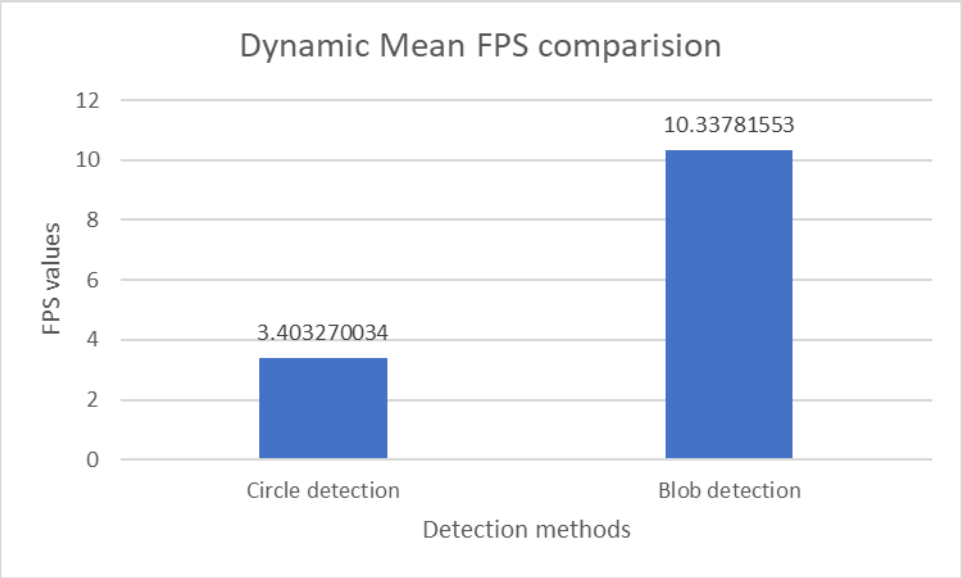


Figure 6.3 - Dynamic mean fps comparison

In dynamic condition, the success rates of detection in two methods are also compared, although the blob detection method didn't give a satisfying success rate, it is still better than the circle detection method as presented in Table 6.1.

Table 6.1 - Detection comparison confidence interval

Detection comparision	Mean norm error	Standard Deviation	Z(0.95)	Confidence interval
blob detection	48.0359	24.65659	1.96	15.28231
circle detection	56.61657	20.89561	1.96	12.95124

The blob detection tends to have a smaller error compared with the circle detection method as shown in Table 6.1.

We couldn't compare the results with Neves' work [50] because they do not provide frame rate, or accuracy. They just highlight that their approach works.

It can be noticed that they use a different vision library (still with Kinect): UAVision computer library.

Nevertheless, we can compare the results with the performance of Rollin' Justin [25] bragging a precision of 2cm in space (our results indicate an average positioning error of 5mm), 5ms in time (our robot takes approximately 1.5 seconds to reach the target after the ball has been dropped/thrown, which is still too slow to be able to catch it), 80% catch rate (our results tend to a 15% catch over 100 ball drops, excluding ball thrown). It can be noticed that there is a significant difference in the material used: custom stereo vision system with high resolution (1600×1200@25Hz) cameras for instance.

Future works, like using of UKF to smooth the trajectory (Kalman filter implementation), and machine learning to improve accuracy (learning as filter for Kinect) can be used in a future implementation of the in-to-net bot.

6.2 Virtual ITNB

6.2.1 Net movement (1D control) and ball motion

In order to validate the behaviour of the virtual ITNB, as explained in section 5.2.2, a sliding net (1D movement) has been designed to simplify the Reinforcement Learning implementation.

The net can follow a random trajectory of the ball seen from a camera POV.

6.2.2 Simple reinforcement learning implementation

The Reinforcement Learning was to be implemented using gym-gazebo software [73] based on OpenAI, but the interface turned out to be very complex for custom robot models and robot environments.

The prototype learning agent in 2D environment (pygame) has therefore been written to validate whether a robot can learn the ball dynamics. It shows a trend that the agent can follow the ball's moving direction and try to catch it. Although in our training, the robot cannot succeed every time but an increasing catching rate needs to be pinpointed.

For the reinforcement learning, in each step, the agent's observation states are robot's current position and ball's current position. The agent's actions are set as velocities at current step.

6.2.3 Reinforcement Learning perspective for Gazebo

The python script for Reinforcement Learning showed great progress in the robot learning of the ball dynamics. The team infers that a simple learning approach can be implemented for the sliding net and a random trajectory of the ball, shown in Figure 5.19. If the learning is successful, it can then be extended to the complete virtual ITNB, shown in Figure 5.2.

Several aspects of the reinforcement learning implementation can be improved, such as:

- Learning parameters to tune (add velocity and other parameters)
- Setup gazebo environment with gym (3D) or ROS nodes (2D)
- Transfer policy from virtual to real robot

7 Conclusion

Inspired from recent research in the field of ball catching robots, a successful ball catching robot has been created, using a Kinect version 2, a Dobot Magician, a custom net designed with Catia V5, and ROS the open-source, meta-operating system, to connect all components (software and hardware).

Even if the performance does not equal the 80% successful catching rate of Rollin' Justin [25], the physical in-to-net bot showed satisfactory results for balls dropped from human height. However, it needs enhancements for balls thrown at it.

In order to improve the results of the physical experiment (ball thrown at the robot), a virtual environment in Gazebo has been created to implement reinforcement learning. A simple one-dimension model of the in-to-net bot has been created (sliding net) which is able to follow the random trajectory of a virtual ball, has been developed. Another model for the complete in-to-net bot has been created in the Gazebo environment, however, the pure reinforcement learning has only been theorised and not implemented, neither on the sliding net, nor on the virtual in-to-net bot. Nevertheless, a simple pygame environment has helped confirmed that a robot agent can learn ball dynamics based on the position of the ball received at specific instants. It may also be enhanced by providing the velocity of the ball to the agent/robot.

Overall, the majority of the objectives have been fulfilled.

Firstly, the ball motion has been converted to a simple entity relative to the robot (a three-dimensional point in the robot workspace in the case of the physical in-to-net bot; a three-dimensional point in the Gazebo environment in the case of the virtual in-to-net bot). Progress can still be made for the physical model, such as increasing the frame rate captured by the Kinect version 2 and speeding up the communication through ROS.

Secondly, the ball motion has been effectively modelled and a simple information (landing point estimation) has been communicated to the robot arm, controlled and able to reach the target. For the physical model, the ROS Service provided

by Dobot company has been used to control the robot. For the virtual model, the Gazebo-ROS control has been used to control the virtual sliding net and the virtual in-to-net bot, which also required the use of Rviz/Moveit!.

Thirdly, according to the literature review, reinforcement learning increases the catching rate of the robot. Even though the reinforcement learning method has only been tested in a pygame environment, the Gazebo environment is ready to be tested with simple learning processes, as well as more complex ones. The impact of reinforcement learning on the catching rate will therefore be confirmed with future implementation. The physical in-to-net bot is also ready to be tested with simple learning processes, thanks to the consideration of the net and the ultrasonic sensor connected to a Raspberry Pi.

Finally, even though the control of the virtual robot has been confirmed by a Gazebo simulation, the machine learning policies have not been implemented. As a consequence, the transfer of learning policies from the virtual environment to the physical environment has not been tested. This step is crucial to validate the impact of reinforcement learning on the catching rate of the physical robot, and it can be carried out in future research after a successful implementation of the reinforcement learning in the simulated environment.

REFERENCES

1. Höfer S. On Decomposability in Robot Reinforcement Learning. 2017. Available at: DOI:http://dx.doi.org/10.14279/depositonce-6054
2. Belousov B., Neumann G., Rothkopf CA., Peters JR. Catching heuristics are optimal control policies. Advances in Neural Information Processing Systems 29. 2016; (Nips): 1426–1434. Available at: DOI:10.1128/AAC.50.4.1612-1613.2006
3. Chapman S. Catching a baseball. American Journal of Physics. : 1968.
4. Christopher John Cornish Hellaby Watkins. Learning from Delayed Rewards. Cambridge University; 1989. Available at: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf (Accessed: 13 March 2019)
5. Corke PI. High-Performance Visual Closed-Loop Robot Control By. 1994; (July).
6. Das S., Das R. Using reinforcement learning to catch a baseball. 2002; : 2808–2812. Available at: DOI:10.1109/icnn.1994.374676
7. Buttazzo GC., Allotta B., Fanizza FP. Mousebuster: a robot for real-time catching. IEEE Control Systems. February 1994; 14(1): 49–56. Available at: DOI:10.1109/37.257894
8. Fernandes DG., Lima PU. A testbed for robotic visual servoing and catching of moving objects. 1998 IEEE International Conference on Electronics, Circuits and Systems. Surfing the Waves of Science and Technology (Cat. No.98EX196). IEEE; pp. 475–478. Available at: DOI:10.1109/ICECS.1998.814924
9. Hong W., Slotine J-JE. Experiments in hand-eye coordination using active vision. Experimental Robotics IV. 2005; : 130–139. Available at: DOI:10.1007/bfb0035204
10. Nagashima K., Inaba M., Inoue H., Nishiwaki K., Ionno A. The humanoid Saika that catches a thrown ball. 2002; : 94–99. Available at: DOI:10.1109/roman.1997.646959
11. Mnih V., Heess N., Graves A., Kavukcuoglu K. Recurrent Models of Visual Attention. 2014; : 1–9. Available at: DOI:ng

12. Lippiello V., Ruggiero F., Siciliano B. 3D monocular robotic ball catching. Robotics and Autonomous Systems. December 2013; 61(12): 1615–1625. Available at: DOI:10.1016/j.robot.2013.06.008
13. Ribnick E., Atev S., Papanikolopoulos NP. Estimating 3D Positions and Velocities of Projectiles from Monocular Views. IEEE Transactions on Pattern Analysis and Machine Intelligence. May 2009; 31(5): 938–944. Available at: DOI:10.1109/TPAMI.2008.247
14. Cigliano P., Lippiello V., Ruggiero F., Siciliano B. Robotic Ball Catching with an Eye-in-Hand Single-Camera System. IEEE Transactions on Control Systems Technology. September 2015; 23(5): 1657–1671. Available at: DOI:10.1109/TCST.2014.2380175
15. McLeod P., Reed N., Dienes Z. The generalized optic acceleration cancellation theory of catching. Journal of Experimental Psychology: Human Perception and Performance. 2006; 32(1): 139–148. Available at: DOI:10.1037/0096-1523.32.1.139
16. Kistemaker DA., Faber H., Beek PJ. Catching fly balls: A simulation study of the Chapman strategy. Human Movement Science. 2009; 28(2): 236–249. Available at: DOI:10.1016/j.humov.2008.11.001
17. Zaal FTJM., Michaels CF. The Information for Catching Fly Balls: Judging and Intercepting Virtual Balls in a CAVE. Journal of Experimental Psychology: Human Perception and Performance. 2003; 29(3): 537–555. Available at: DOI:10.1037/0096-1523.29.3.537
18. Nemec B., Zorko M., Zlajpah L. Learning of a ball-in-a-cup playing robot. 19th International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD 2010). IEEE; 2010. pp. 297–301. Available at: DOI:10.1109/RAAD.2010.5524570
19. Flash T., Hochner B. Motor primitives in vertebrates and invertebrates. Current Opinion in Neurobiology. December 2005; 15(6): 660–666. Available at: DOI:10.1016/j.conb.2005.10.011
20. Kober J., Peters J. Policy search for motor primitives in robotics. Machine Learning. 6 July 2011; 84(1–2): 171–203. Available at: DOI:10.1007/s10994-010-5223-6

21. Peters J., Schaal S. Reinforcement Learning for Parameterized Motor Primitives. The 2006 IEEE International Joint Conference on Neural Network Proceedings. IEEE; 2006. pp. 73–80. Available at: DOI:10.1109/IJCNN.2006.246662
22. Ijspeert AJ., Nakanishi J., Schaal S. Movement imitation with nonlinear dynamical systems in humanoid robots. Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292). IEEE; pp. 1398–1403. Available at: DOI:10.1109/ROBOT.2002.1014739
23. Kim S., Shukla A., Billard A. Catching objects in flight. IEEE Transactions on Robotics. IEEE; 2014; 30(5): 1049–1065. Available at: DOI:10.1109/TRO.2014.2316022
24. Bäuml B., Wimböck T., Hirzinger G. Kinematically optimal catching a flying ball with a hand-arm-system. IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings. 2010; : 2592–2599. Available at: DOI:10.1109/IROS.2010.5651175
25. Bäuml B., Schmidt F., Wimböck T., Birbach O., Dietrich A., Fuchs M., et al. Catching flying balls and preparing coffee: Humanoid Rollin' Justin performs dynamic and sensitive tasks. Proceedings - IEEE International Conference on Robotics and Automation. 2011; (1): 3443–3444. Available at: DOI:10.1109/ICRA.2011.5980073
26. Birbach O., Frese U., Bäuml B. Realtime Perception for Catching a Flying Ball with a Mobile Humanoid. Proceedings - IEEE International Conference on Robotics and Automation. 2011; : 5955–5962. Available at: DOI:10.1109/ICRA.2011.5980138
27. Birbach O. Tracking and Calibration for a Ball Catching Humanoid Robot. 2012; 3(November).
28. Sensors - Wiki for iCub and Friends. Available at: http://wiki.icub.org/wiki/Sensors (Accessed: 13 March 2019)
29. Kober J., Peters J. Imitation and Reinforcement Learning. IEEE Robotics & Automation Magazine. June 2010; 17(2): 55–62. Available at: DOI:10.1109/MRA.2010.936952

30. Kober J., Glisson M., Mistry M. Playing catch and juggling with a humanoid robot. 2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012). IEEE; 2012. pp. 875–881. Available at: DOI:10.1109/HUMANOIDS.2012.6651623
31. Zeng A., Song S., Lee J., Rodriguez A., Funkhouser T. TossingBot: Learning to Throw Arbitrary Objects with Residual Physics. 27 March 2019; Available at: http://arxiv.org/abs/1903.11239
32. Hock O., Šedo J. Forward and Inverse Kinematics Using Pseudoinverse and Transposition Method for Robotic Arm DOBOT. Kinematics. 2017; Available at: DOI:10.5772/intechopen.71417
33. DOBOT. Dobot Magician Specifications and Shipping List. 2019. Available at: https://www.dobot.cc/dobot-magician/specification.html (Accessed: 21 March 2019)
34. Windows MK for. Interface Guidelines. Human interface Guidelines. 2013; v1.8: 1–142.
35. Raspberry Pi Foundation. Raspberry Pi 3 Model B. 2019. Available at: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ (Accessed: 21 March 2019)
36. Documentation - ROS Wiki. Available at: http://wiki.ros.org/ (Accessed: 12 March 2019)
37. Open Source Robotics Foundation. kinetic/Installation - ROS Wiki. 2018. Available at: http://wiki.ros.org/kinetic/Installation (Accessed: 20 March 2019)
38. Open Source Robotics Foundation. kinetic/Installation/Ubuntu - ROS Wiki. 2017. Available at: http://wiki.ros.org/kinetic/Installation/Ubuntu (Accessed: 20 March 2019)
39. Ling X., Zhao Y., Gong L., Liu C., Wang T. Dual-arm cooperation and implementing for robotic harvesting tomato using binocular vision. Robotics and Autonomous Systems. April 2019; 114: 134–143. Available at: DOI:10.1016/j.robot.2019.01.019 (Accessed: 20 March 2019)
40. ModMyPi LTD. HC-SR04 Ultrasonic Range Sensor on the Raspberry Pi. 2014. Available at: https://www.modmypi.com/blog/hc-sr04-ultrasonic-

range-sensor-on-the-raspberry-pi (Accessed: 20 March 2019)

41. Not Enough Tech. Connect Raspberry Pi to laptop PC in 4 simple steps. 2016. Available at: https://notenoughtech.com/raspberry-pi/connect-raspberrypi-pc/ (Accessed: 20 March 2019)
42. Open Source Robotics Foundation. simulator_gazebo/SystemRequirements - ROS Wiki. Available at: http://wiki.ros.org/simulator_gazebo/SystemRequirements (Accessed: 21 March 2019)
43. Dobot. Dobot Magician Specifications and Shipping List. 2019. Available at: https://www.dobot.cc/dobot-magician/specification.html (Accessed: 21 April 2019)
44. Shenzhen Yuejiang Technology Co. L. Dobot Magician User Manual. 2017; : 138.
45. Granja M., Chang N., Granja V., Duque M., Llulluna F. Comparison between Standard and Modified Denavit-Hartenberg Methods in Robotics Modelling. Proceedings of the 2nd World Congress on Mechanical, Chemical, and Material Engineering. 2016; 1(1): 1–10. Available at: DOI:10.11159/icmie16.118
46. Qassem MA., Abuhadrous I., Elaydi H. Modeling and simulation of 5 DOF educational robot arm. Proceedings - 2nd IEEE International Conference on Advanced Computer Control, ICACC 2010. 2010; 5(April 2015): 569–574. Available at: DOI:10.1109/ICACC.2010.5487136
47. Dobot.cc. [OFFICIAL]Dobot Download Center. 2019. Available at: https://www.dobot.cc/downloadcenter.html?sub_cat=72#sub-download (Accessed: 9 May 2019)
48. Henriques JF., Caseiro R., Martins P., Batista J. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE 1 High-Speed Tracking with Kernelized Correlation Filters. Available at: http://www.robots.ox.ac.uk/~joao/publications/henriques_tpami2015.pdf (Accessed: 9 May 2019)
49. Opencv Community. OpenCV: cv::TrackerKCF Class Reference. 2019. Available at:

https://docs.opencv.org/3.4/d2/dff/classcv_1_1TrackerKCF.html (Accessed: 9 May 2019)

50. Neves AJR., Trifan A., Dias P., Azevedo JL. Detection of Aerial Balls in Robotic Soccer Using a Mixture of Color and Depth Information. 2015; Available at: DOI:10.1109/ICARSC.2015.13 (Accessed: 11 April 2019)
51. OpenCV community. OpenCV: Thresholding Operations using inRange. 2018. Available at: https://docs.opencv.org/3.4.3/da/d97/tutorial_threshold_inRange.html (Accessed: 9 May 2019)
52. opencv dev team. Hough Circle Transform — OpenCV 2.4.13.7 documentation. 2019. Available at: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html (Accessed: 9 May 2019)
53. Opencv Dev Team. OpenCV: cv::SimpleBlobDetector Class Reference. 2019. Available at: https://docs.opencv.org/3.4.3/d0/d7a/classcv_1_1SimpleBlobDetector.html (Accessed: 9 May 2019)
54. OpenKinect community. OpenKinect. 2012. Available at: https://openkinect.org/wiki/Main_Page (Accessed: 16 April 2019)
55. Owen JP., Ryu WS. The effects of linear and quadratic drag on falling spheres: An undergraduate laboratory. European Journal of Physics. 2005; 26(6): 1085–1091. Available at: DOI:10.1088/0143-0807/26/6/016
56. Yang H., Fan M., Liu A., Dong L. General formulas for drag coefficient and settling velocity of sphere based on theoretical law. International Journal of Mining Science and Technology. China University of Mining & Technology; 2015; 25(2): 219–223. Available at: DOI:10.1016/j.ijmst.2015.02.009
57. Scipy.org. scipy.integrate.odeint. 2019. Available at: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html (Accessed: 26 March 2019)
58. Belgacem CH. Analysis of projectile motion with quadratic air resistance from a nonzero height using the Lambert W function . Journal of Taibah University for Science. Taibah University; 2016; 11(2): 328–331. Available

at: DOI:10.1016/j.jtusci.2016.02.009

59. Joseph L., Cacace J. Mastering ROS for robotics programming : design, build, and simulate complex robots using Robot Operating System.
60. P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli GR. MeshLab: an Open-Source Mesh Processing Tool. Sixth Eurographics Italian Chapter Conference. 2008; : 129–136. Available at: DOI:10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136
61. Blender. Home of the Blender project. 2019. Available at: https://www.blender.org/ (Accessed: 9 April 2019)
62. Open Source Robotics Foundation. Gazebo Tutorial : URDF in Gazebo. 2014. Available at: http://gazebosim.org/tutorials/?tut=ros_urdf (Accessed: 21 April 2019)
63. Open Source Robotics Foundation. Gazebo Tutorial : ROS control. 2014. Available at: http://gazebosim.org/tutorials/?tut=ros_control (Accessed: 10 April 2019)
64. Open Source Robotics Foundation. rqt - ROS Wiki. 2016. Available at: http://wiki.ros.org/rqt (Accessed: 10 April 2019)
65. rviz - ROS Wiki. 2018. Available at: http://wiki.ros.org/rviz (Accessed: 10 May 2019)
66. Ioan A. Sucan and Sachin Chitta. MoveIt. 2019. Available at: http://moveit.ros.org (Accessed: 2 April 2019)
67. Open Source Robotics Foundation. Animated Box. 2014. Available at: http://gazebosim.org/tutorials?tut=animated_box (Accessed: 2 May 2019)
68. Pygame community. About - pygame wiki. 2018. Available at: https://www.pygame.org/wiki/about (Accessed: 10 May 2019)
69. Wu Y., Mansimov E., Liao S., Grosse R., Ba J. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation.
70. Konda VR., Tsitsiklis JN. Actor-Critic Algorithms.
71. Chen J., Chao L. Positioning error analysis for robot manipulators with all rotary joints. Proceedings. 1986 IEEE International Conference on Robotics and Automation. Institute of Electrical and Electronics Engineers;

pp. 1011–1016. Available at: DOI:10.1109/ROBOT.1986.1087544 (Accessed: 10 May 2019)

72. Dobot. Dobot Magician Specifications and Shipping List | DOBOT. 2019. Available at: https://www.dobot.cc/dobot-magician/specification.html (Accessed: 10 May 2019)
73. Zamora I., Gonzalez Lopez N., Mayoral Vilches V., Cordero AH., Robotics E. Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo. Available at: http://erlerobotics.com/whitepaper/robot_gym.pdf (Accessed: 26 March 2019)

APPENDICES

Appendix A – Inverse kinematics of Dobot Magician

The indications given to the robot to reach a specific position T with a specific orientation Q are,

- T = $[x_t, y_t, z_t]$ (the target point)
- ψ (angle between x_1 and link {5}) to define the orientation Q

From Figure A.1, θ_1 can be derived,

$$\theta_1 = Atan2(y_t, x_t)$$

Another relation can be derived from Figure A.1,

$$r_t = \sqrt{(x_t)^2 + (y_t)^2}$$

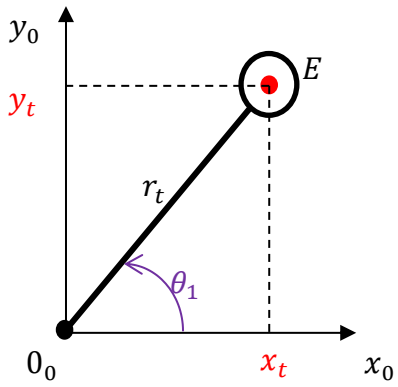


Figure A.1 - Dobot top view

The relationship between the orientation angle and the joint angles is,

$$\psi = \theta_2 + \theta_3 + \theta_4$$

The coordinates of the wrist joint (0_3) can be determined,

$$r_4 = r_t - a_4 \cos(\psi)$$

$$z_4 = z_t - a_4 \sin(\psi)$$

The angles in the triangle $0_1 0_2 0_3$ can then be found,

$$\alpha = Atan2(z_4 - d_1, r_4)$$

The distance $0_1 0_3$ is,

$$s = \sqrt{(z_4 - d_1)^2 + (r_4)^2}$$

Using the law of cosines in $0_1 0_2 0_3$,

$$\beta = \frac{s^2 + {a_2}^2 - {a_3}^2}{2 * a_2 * s}; \quad \sin(\beta) = \pm\sqrt{1 - \cos(\beta)^2}; \quad \beta = Atan2(\sin(\beta), \cos(\beta))$$

θ_2 can then be determined,

$$\theta_2 = \alpha \pm \beta$$

In order to determine θ_3, the following expressions are used,

$$r_4 = r_t - a_4\cos(\psi) = a_2\cos(\theta_2) + a_3\cos(\theta_2 + \theta_3)$$

$$z_4 = z_t - a_4\sin(\psi) = a_2\sin(\theta_2) + a_3\sin(\theta_2 + \theta_3) + d_1$$

$$s = \sqrt{(z_4 - d_1)^2 + (r_4)^2}$$

θ_3 can be derived,

$$\cos(\theta_3) = \frac{s^2 - {a_2}^2 - {a_3}^2}{2 * a_2 * a_3};\quad \sin(\theta_3) = \pm\sqrt{1 - \cos(\theta_3)^2};\quad \theta_3 = Atan2(\sin(\theta_3), \cos(\theta_3))$$

According to Figure A.2,

$$\theta_3 = \pm Atan2(\sin(\theta_3), \cos(\theta_3))$$

Finally,

$$\theta_4 = \psi - \theta_2 - \theta_3$$

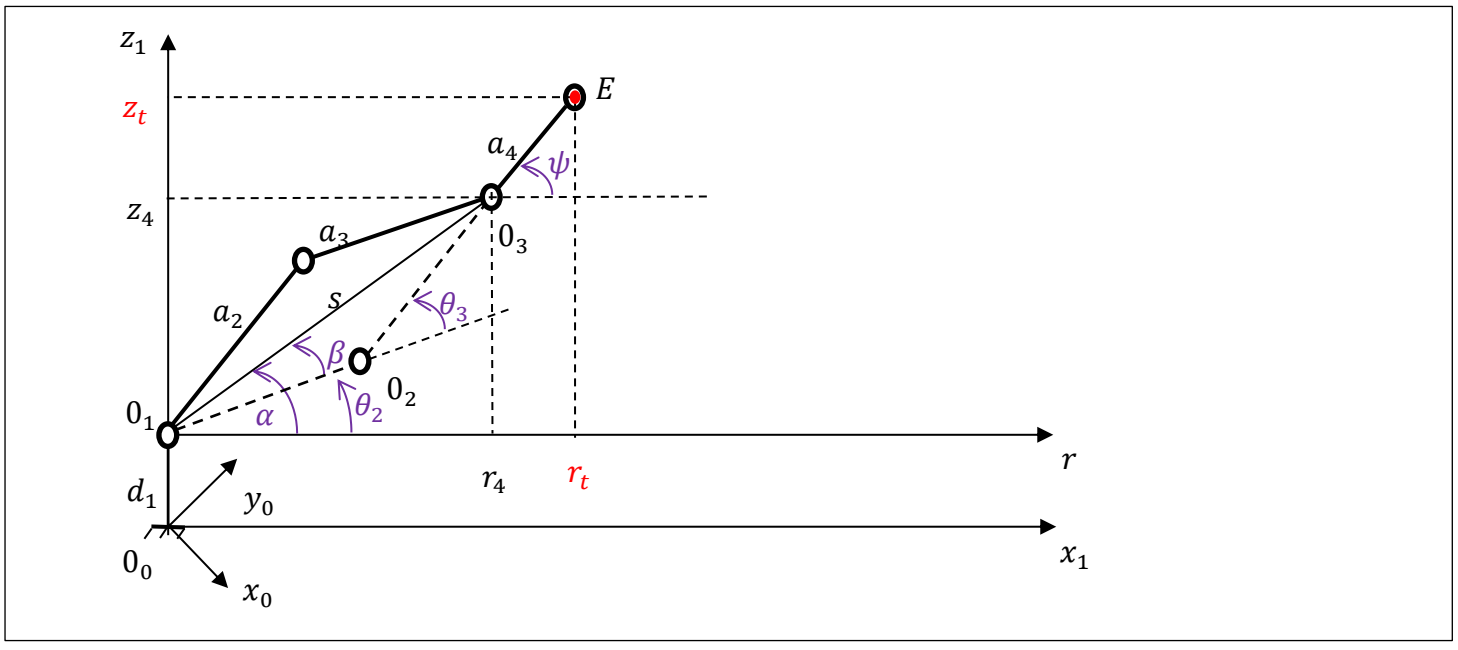


Figure A.2 - Dobot side view

The solutions of the inverse kinematics are gathered in Table A.**1**.

Table A.1 - Solutions of the inverse kinematics (Dobot)

First solution	Second solution
θ_1	θ_{1_bis}
$\theta_2 = \alpha - \beta$	$\theta_{2\_bis} = \alpha + \beta$
θ_3	$\theta_{3\_bis} = -\theta_3$
$\theta_4 = \psi - \theta_2 - \theta_3$	$\theta_{4\_bis} = \psi - \theta_{2_{bis}} - \theta_{3\_bis}$

According to the tests performed on the Dobot Magician, the angle ψ remains constant (ψ = 0), always parallel to the ground (it is not controlled, it is mechanically maintained to this position).

Appendix B – Risk assessment

The risk assessment for the project is shown in Table B.1.

The risk rating is based on the following:

- 1 = Negligible
- 2 = Minor
- 3 = Medium
- 4 = Major
- 5 = Severe

Table B.1 - Risk assessment

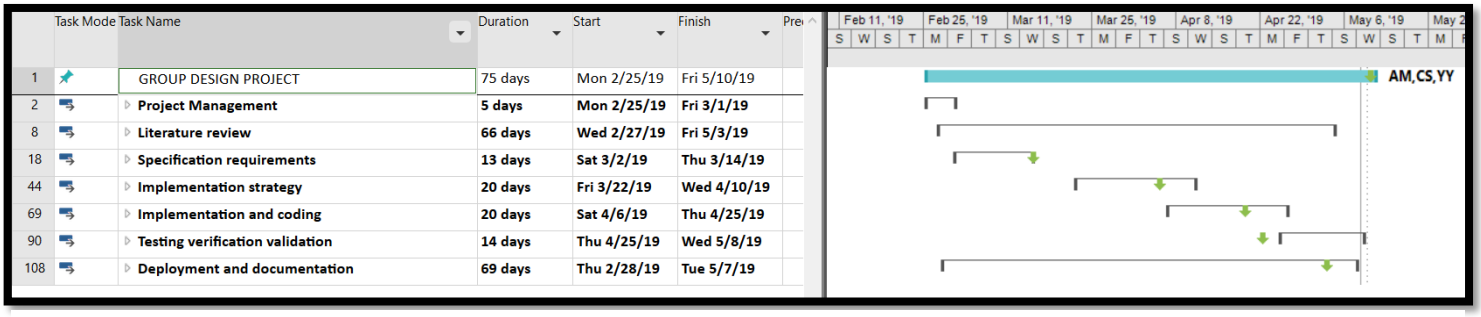
Task / event	**Significant Hazards**	**Who is affected**	**Existing controls**	**Risk rating**			**Additional control needed?**
		Human / Robot	None / describe	Consequence	Likelihood	Total = C x L	Risk mitigation
Robot testing	Break robot components	R	No spare components, simulation	3	1	3	Get new components (fast)
	Injury the human	R	Safety covers, health and safety rules	1	1	1	N/A
	Brightness (blinding sensors)	R /P	Light controlled testing room	1	1	1	N/A
Software implementation	Compatibility issues (platforms, compilation, language, updates)	R	Thorough literature review	2	2	4	N/A

	Poor documentation (programs)	P		2	1	2	N/A
	Sensor failure (outrange, outbounds, noise)	R	Robust algorithm	1	3	3	N/A
	Output malfunction (arm moving not as expected)	H / R	Boundary conditions	2	2	4	N/A
	Processing unit overflow (ML related, memory)	R	Preliminary memory estimation	2	2	4	N/A
Hardware Implementation	Electromechanical failure / damage (wires, components, overheating),	R	Regular maintenance	3	1	3	N/A
	Loose wire	R	cable ties	2	1	2	N/A
	Overheating components	R	Regular maintenance, cooling fan.	2	1	2	N/A
	Missing parts	R	Spares	1	2	2	N/A
	Faulty parts	R	Return policy	2	1	2	Buy a new one
	Mechanical interface failure	R	Preliminary design and maintenance	2	1	2	Redesign
Inventory stock supervision	Unavailable components / technology	P	Find alternatives	1	1	1	N/A
	Lack of resources	P		1	1	1	N/A

Budgeting	Lack of funds to get components/ software	P	Find cheap alternatives	2	1	2	N/A
	Unexpected rise in cost	P		2	1	2	N/A
Scheduling	Go beyond deadline	P	Respect Gantt	2	1	2	N/A
	Low efficiency of resource allocation and organisation	P	Motivate team members	3	1	3	N/A
	Unrealistic project plan schedule	P	Update Gantt	2	1	2	N/A
	Delayed meetings	P	Inform about delays	1	1	1	N/A
Organization Management	Unexpected absence / Long absence	P	Supervisors advice	1	1	1	N/A
	Poor/lack/ error of Communication (supervisors and/or students)	P	Supervisors advice	1	1	1	N/A

Appendix C – Time management

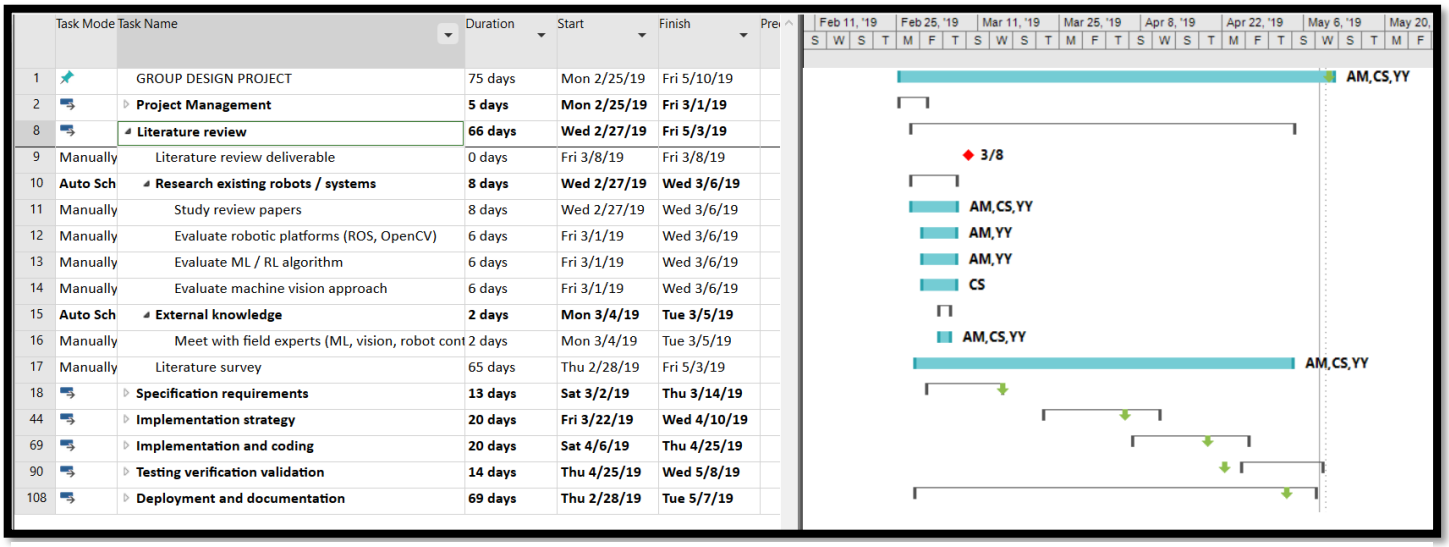
C.1 Overview of the main tasks



	Task Mode	Task Name	Duration	Start	Finish	Pre
1		GROUP DESIGN PROJECT	75 days	Mon 2/25/19	Fri 5/10/19	
2		**Project Management**	**5 days**	**Mon 2/25/19**	**Fri 3/1/19**	
8		**Literature review**	**66 days**	**Wed 2/27/19**	**Fri 5/3/19**	
18		**Specification requirements**	**13 days**	**Sat 3/2/19**	**Thu 3/14/19**	
44		**Implementation strategy**	**20 days**	**Fri 3/22/19**	**Wed 4/10/19**	
69		**Implementation and coding**	**20 days**	**Sat 4/6/19**	**Thu 4/25/19**	
90		**Testing verification validation**	**14 days**	**Thu 4/25/19**	**Wed 5/8/19**	
108		**Deployment and documentation**	**69 days**	**Thu 2/28/19**	**Tue 5/7/19**	

Figure C.1 - Overview Gantt

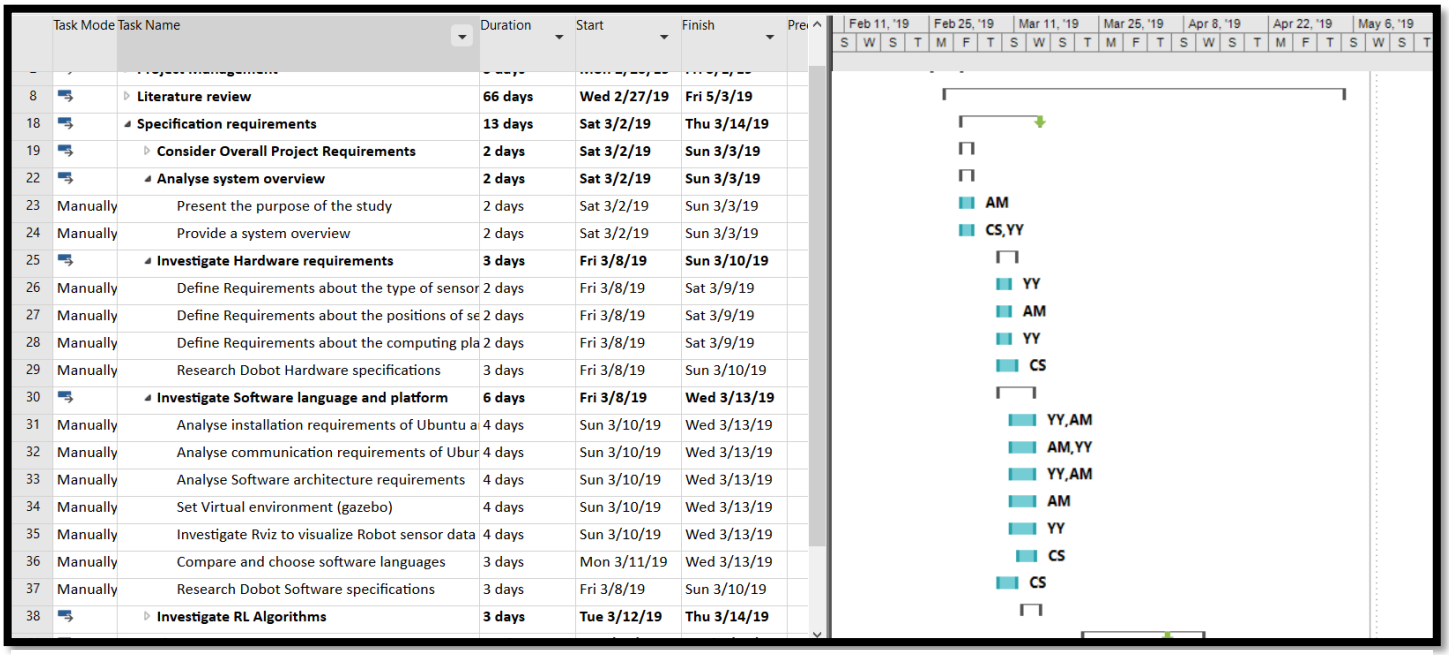
C.2 Literature review



	Task Mode	Task Name	Duration	Start	Finish	Pre
1		GROUP DESIGN PROJECT	75 days	Mon 2/25/19	Fri 5/10/19	
2		**Project Management**	**5 days**	**Mon 2/25/19**	**Fri 3/1/19**	
8		**Literature review**	**66 days**	**Wed 2/27/19**	**Fri 5/3/19**	
9	Manually	Literature review deliverable	0 days	Fri 3/8/19	Fri 3/8/19	
10	**Auto Sch**	**Research existing robots / systems**	**8 days**	**Wed 2/27/19**	**Wed 3/6/19**	
11	Manually	Study review papers	8 days	Wed 2/27/19	Wed 3/6/19	
12	Manually	Evaluate robotic platforms (ROS, OpenCV)	6 days	Fri 3/1/19	Wed 3/6/19	
13	Manually	Evaluate ML / RL algorithm	6 days	Fri 3/1/19	Wed 3/6/19	
14	Manually	Evaluate machine vision approach	6 days	Fri 3/1/19	Wed 3/6/19	
15	**Auto Sch**	**External knowledge**	**2 days**	**Mon 3/4/19**	**Tue 3/5/19**	
16	Manually	Meet with field experts (ML, vision, robot cont	2 days	Mon 3/4/19	Tue 3/5/19	
17	Manually	Literature survey	65 days	Thu 2/28/19	Fri 5/3/19	
18		**Specification requirements**	**13 days**	**Sat 3/2/19**	**Thu 3/14/19**	
44		**Implementation strategy**	**20 days**	**Fri 3/22/19**	**Wed 4/10/19**	
69		**Implementation and coding**	**20 days**	**Sat 4/6/19**	**Thu 4/25/19**	
90		**Testing verification validation**	**14 days**	**Thu 4/25/19**	**Wed 5/8/19**	
108		**Deployment and documentation**	**69 days**	**Thu 2/28/19**	**Tue 5/7/19**	

Figure C.2 - Literature review Gantt

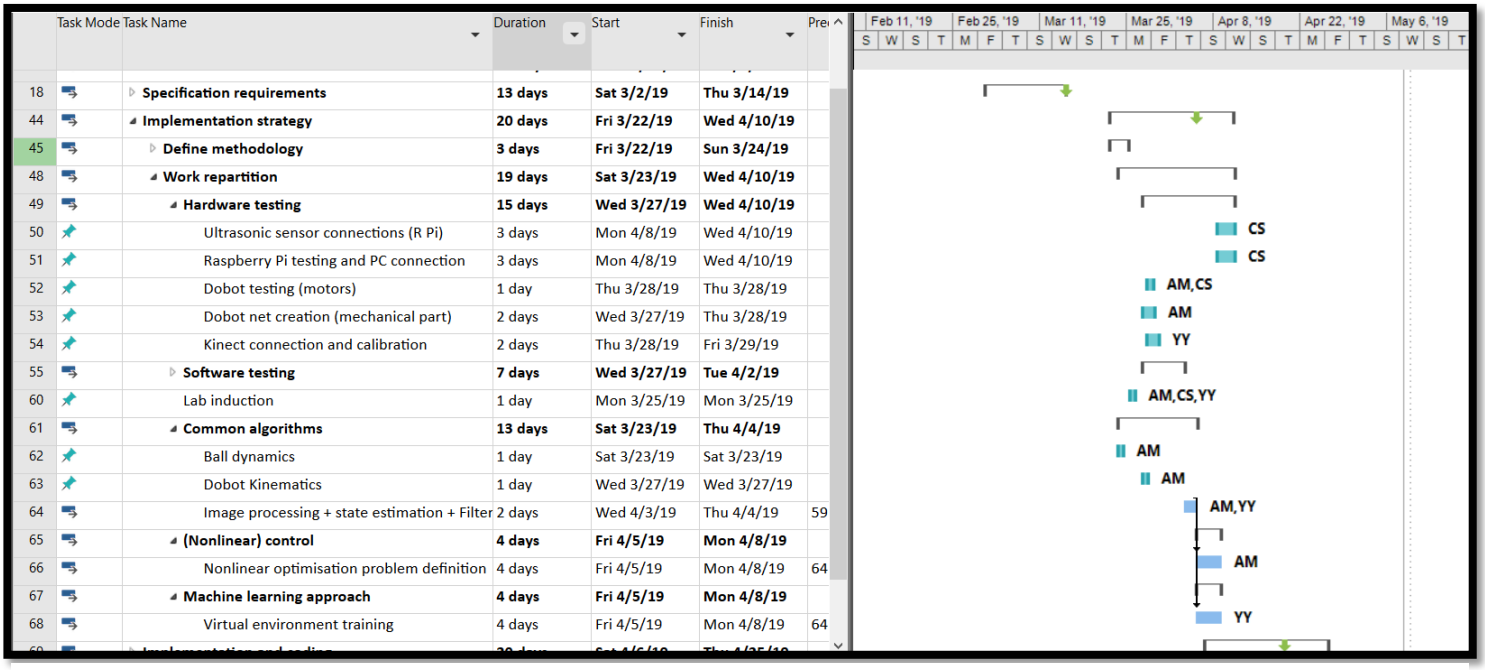
C.3 Specification requirements



	Task Mode	Task Name	Duration	Start	Finish	Pre
8		**Literature review**	**66 days**	**Wed 2/27/19**	**Fri 5/3/19**	
18		**Specification requirements**	**13 days**	**Sat 3/2/19**	**Thu 3/14/19**	
19		**Consider Overall Project Requirements**	**2 days**	**Sat 3/2/19**	**Sun 3/3/19**	
22		**Analyse system overview**	**2 days**	**Sat 3/2/19**	**Sun 3/3/19**	
23	Manually	Present the purpose of the study	2 days	Sat 3/2/19	Sun 3/3/19	
24	Manually	Provide a system overview	2 days	Sat 3/2/19	Sun 3/3/19	
25		**Investigate Hardware requirements**	**3 days**	**Fri 3/8/19**	**Sun 3/10/19**	
26	Manually	Define Requirements about the type of sensor	2 days	Fri 3/8/19	Sat 3/9/19	
27	Manually	Define Requirements about the positions of se	2 days	Fri 3/8/19	Sat 3/9/19	
28	Manually	Define Requirements about the computing pla	2 days	Fri 3/8/19	Sat 3/9/19	
29	Manually	Research Dobot Hardware specifications	3 days	Fri 3/8/19	Sun 3/10/19	
30		**Investigate Software language and platform**	**6 days**	**Fri 3/8/19**	**Wed 3/13/19**	
31	Manually	Analyse installation requirements of Ubuntu a	4 days	Sun 3/10/19	Wed 3/13/19	
32	Manually	Analyse communication requirements of Ubur	4 days	Sun 3/10/19	Wed 3/13/19	
33	Manually	Analyse Software architecture requirements	4 days	Sun 3/10/19	Wed 3/13/19	
34	Manually	Set Virtual environment (gazebo)	4 days	Sun 3/10/19	Wed 3/13/19	
35	Manually	Investigate Rviz to visualize Robot sensor data	4 days	Sun 3/10/19	Wed 3/13/19	
36	Manually	Compare and choose software languages	3 days	Mon 3/11/19	Wed 3/13/19	
37	Manually	Research Dobot Software specifications	3 days	Fri 3/8/19	Sun 3/10/19	
38		**Investigate RL Algorithms**	**3 days**	**Tue 3/12/19**	**Thu 3/14/19**	

Figure C.3 - Specification requirements Gantt

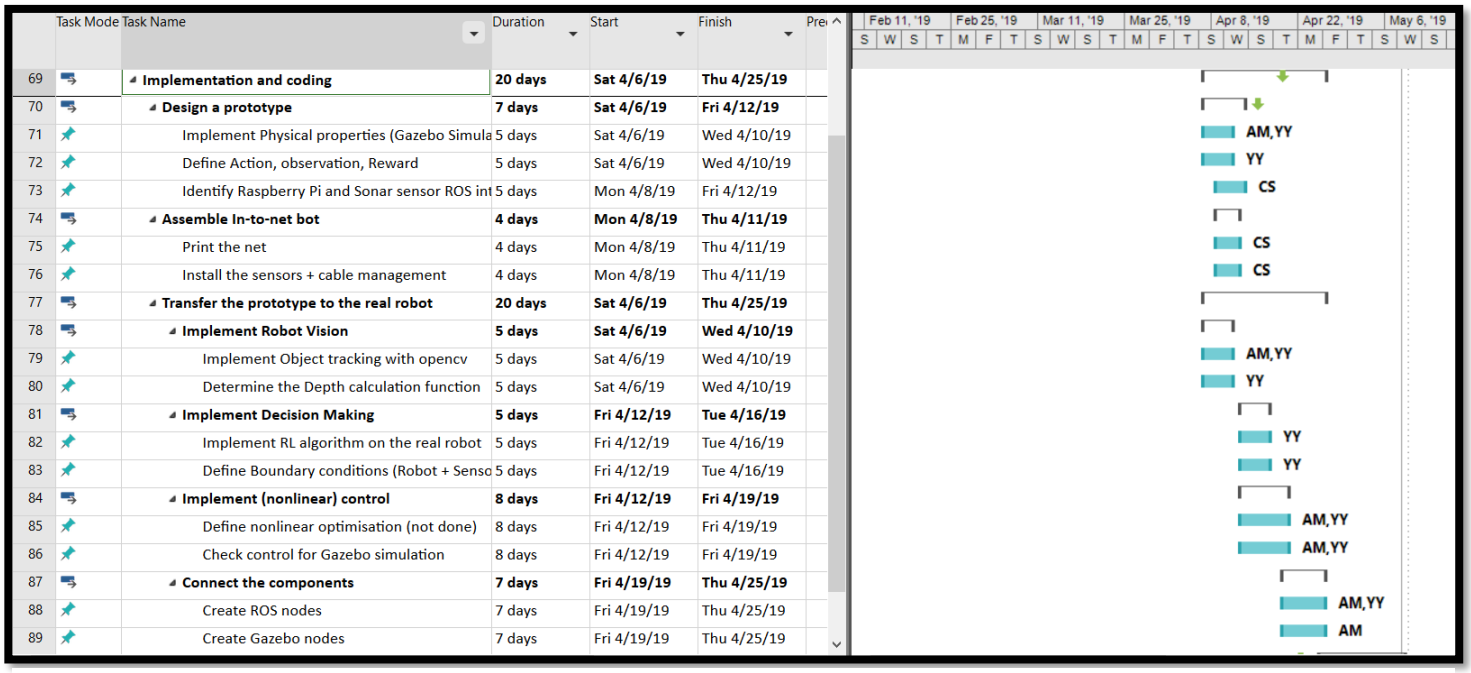
C.4 Implementation strategy



	Task Mode	Task Name	Duration	Start	Finish	Pre
18		**Specification requirements**	**13 days**	**Sat 3/2/19**	**Thu 3/14/19**	
44		**Implementation strategy**	**20 days**	**Fri 3/22/19**	**Wed 4/10/19**	
45		**Define methodology**	**3 days**	**Fri 3/22/19**	**Sun 3/24/19**	
48		**Work repartition**	**19 days**	**Sat 3/23/19**	**Wed 4/10/19**	
49		**Hardware testing**	**15 days**	**Wed 3/27/19**	**Wed 4/10/19**	
50		Ultrasonic sensor connections (R Pi)	3 days	Mon 4/8/19	Wed 4/10/19	
51		Raspberry Pi testing and PC connection	3 days	Mon 4/8/19	Wed 4/10/19	
52		Dobot testing (motors)	1 day	Thu 3/28/19	Thu 3/28/19	
53		Dobot net creation (mechanical part)	2 days	Wed 3/27/19	Thu 3/28/19	
54		Kinect connection and calibration	2 days	Thu 3/28/19	Fri 3/29/19	
55		**Software testing**	**7 days**	**Wed 3/27/19**	**Tue 4/2/19**	
60		Lab induction	1 day	Mon 3/25/19	Mon 3/25/19	
61		**Common algorithms**	**13 days**	**Sat 3/23/19**	**Thu 4/4/19**	
62		Ball dynamics	1 day	Sat 3/23/19	Sat 3/23/19	
63		Dobot Kinematics	1 day	Wed 3/27/19	Wed 3/27/19	
64		Image processing + state estimation + Filter	2 days	Wed 4/3/19	Thu 4/4/19	59
65		**(Nonlinear) control**	**4 days**	**Fri 4/5/19**	**Mon 4/8/19**	
66		Nonlinear optimisation problem definition	4 days	Fri 4/5/19	Mon 4/8/19	64
67		**Machine learning approach**	**4 days**	**Fri 4/5/19**	**Mon 4/8/19**	
68		Virtual environment training	4 days	Fri 4/5/19	Mon 4/8/19	64

Figure C.4 - Implementation strategy Gantt

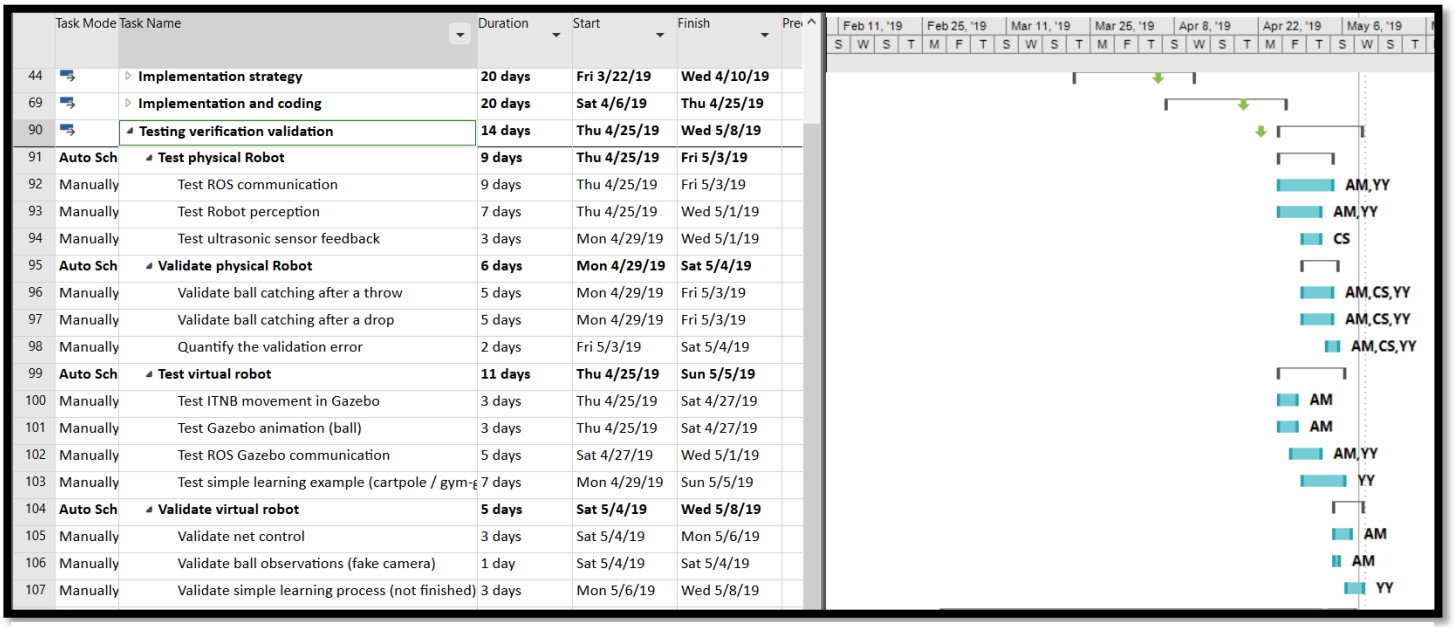
C.5 Implementation and coding



	Task Mode	Task Name	Duration	Start	Finish	Pre
69		**Implementation and coding**	**20 days**	**Sat 4/6/19**	**Thu 4/25/19**	
70		**Design a prototype**	**7 days**	**Sat 4/6/19**	**Fri 4/12/19**	
71		Implement Physical properties (Gazebo Simula	5 days	Sat 4/6/19	Wed 4/10/19	
72		Define Action, observation, Reward	5 days	Sat 4/6/19	Wed 4/10/19	
73		Identify Raspberry Pi and Sonar sensor ROS int	5 days	Mon 4/8/19	Fri 4/12/19	
74		**Assemble In-to-net bot**	**4 days**	**Mon 4/8/19**	**Thu 4/11/19**	
75		Print the net	4 days	Mon 4/8/19	Thu 4/11/19	
76		Install the sensors + cable management	4 days	Mon 4/8/19	Thu 4/11/19	
77		**Transfer the prototype to the real robot**	**20 days**	**Sat 4/6/19**	**Thu 4/25/19**	
78		**Implement Robot Vision**	**5 days**	**Sat 4/6/19**	**Wed 4/10/19**	
79		Implement Object tracking with opencv	5 days	Sat 4/6/19	Wed 4/10/19	
80		Determine the Depth calculation function	5 days	Sat 4/6/19	Wed 4/10/19	
81		**Implement Decision Making**	**5 days**	**Fri 4/12/19**	**Tue 4/16/19**	
82		Implement RL algorithm on the real robot	5 days	Fri 4/12/19	Tue 4/16/19	
83		Define Boundary conditions (Robot + Senso	5 days	Fri 4/12/19	Tue 4/16/19	
84		**Implement (nonlinear) control**	**8 days**	**Fri 4/12/19**	**Fri 4/19/19**	
85		Define nonlinear optimisation (not done)	8 days	Fri 4/12/19	Fri 4/19/19	
86		Check control for Gazebo simulation	8 days	Fri 4/12/19	Fri 4/19/19	
87		**Connect the components**	**7 days**	**Fri 4/19/19**	**Thu 4/25/19**	
88		Create ROS nodes	7 days	Fri 4/19/19	Thu 4/25/19	
89		Create Gazebo nodes	7 days	Fri 4/19/19	Thu 4/25/19	

Figure C.5 - Implementation and coding Gantt

C.6 Testing verification validation



	Task Mode	Task Name	Duration	Start	Finish	Pre
44		**Implementation strategy**	**20 days**	**Fri 3/22/19**	**Wed 4/10/19**	
69		**Implementation and coding**	**20 days**	**Sat 4/6/19**	**Thu 4/25/19**	
90		**Testing verification validation**	**14 days**	**Thu 4/25/19**	**Wed 5/8/19**	
91	**Auto Sch**	**Test physical Robot**	**9 days**	**Thu 4/25/19**	**Fri 5/3/19**	
92	Manually	Test ROS communication	9 days	Thu 4/25/19	Fri 5/3/19	
93	Manually	Test Robot perception	7 days	Thu 4/25/19	Wed 5/1/19	
94	Manually	Test ultrasonic sensor feedback	3 days	Mon 4/29/19	Wed 5/1/19	
95	**Auto Sch**	**Validate physical Robot**	**6 days**	**Mon 4/29/19**	**Sat 5/4/19**	
96	Manually	Validate ball catching after a throw	5 days	Mon 4/29/19	Fri 5/3/19	
97	Manually	Validate ball catching after a drop	5 days	Mon 4/29/19	Fri 5/3/19	
98	Manually	Quantify the validation error	2 days	Fri 5/3/19	Sat 5/4/19	
99	**Auto Sch**	**Test virtual robot**	**11 days**	**Thu 4/25/19**	**Sun 5/5/19**	
100	Manually	Test ITNB movement in Gazebo	3 days	Thu 4/25/19	Sat 4/27/19	
101	Manually	Test Gazebo animation (ball)	3 days	Thu 4/25/19	Sat 4/27/19	
102	Manually	Test ROS Gazebo communication	5 days	Sat 4/27/19	Wed 5/1/19	
103	Manually	Test simple learning example (cartpole / gym-g	7 days	Mon 4/29/19	Sun 5/5/19	
104	**Auto Sch**	**Validate virtual robot**	**5 days**	**Sat 5/4/19**	**Wed 5/8/19**	
105	Manually	Validate net control	3 days	Sat 5/4/19	Mon 5/6/19	
106	Manually	Validate ball observations (fake camera)	1 day	Sat 5/4/19	Sat 5/4/19	
107	Manually	Validate simple learning process (not finished)	3 days	Mon 5/6/19	Wed 5/8/19	

Figure C.6 - Testing verification validation Gantt

C.7 Deployment and documentation

	Task Mode	Task Name	Duration	Start	Finish	Pre
1		GROUP DESIGN PROJECT	75 days	Mon 2/25/19	Fri 5/10/19	
2		**Project Management**	**5 days**	**Mon 2/25/19**	**Fri 3/1/19**	
8		**Literature review**	**66 days**	**Wed 2/27/19**	**Fri 5/3/19**	
18		**Specification requirements**	**13 days**	**Sat 3/2/19**	**Thu 3/14/19**	
44		**Implementation strategy**	**20 days**	**Fri 3/22/19**	**Wed 4/10/19**	
69		**Implementation and coding**	**20 days**	**Sat 4/6/19**	**Thu 4/25/19**	
90		**Testing verification validation**	**14 days**	**Thu 4/25/19**	**Wed 5/8/19**	
108		**Deployment and documentation**	**69 days**	**Thu 2/28/19**	**Tue 5/7/19**	
109	**Auto Sch**	**Write the deliverables**	**65 days**	**Thu 2/28/19**	**Fri 5/3/19**	
117	**Auto Sch**	**Investigate physical ITNB**	**6 days**	**Mon 4/29/19**	**Sat 5/4/19**	
118	Manually	Vision system improvement	5 days	Mon 4/29/19	Fri 5/3/19	
119	Manually	Control system improvement	1 day	Mon 4/29/19	Mon 4/29/19	
120	Manually	Final check physical ITNB	2 days	Fri 5/3/19	Sat 5/4/19	
121	**Auto Sch**	**Investigate virtual ITNB**	**4 days**	**Sat 5/4/19**	**Tue 5/7/19**	
122	Manually	ROS Control improvement	3 days	Sat 5/4/19	Mon 5/6/19	
123	Manually	Ball trajectory improvement	1 day	Sat 5/4/19	Sat 5/4/19	
124	Manually	RL process improvement	3 days	Sun 5/5/19	Tue 5/7/19	

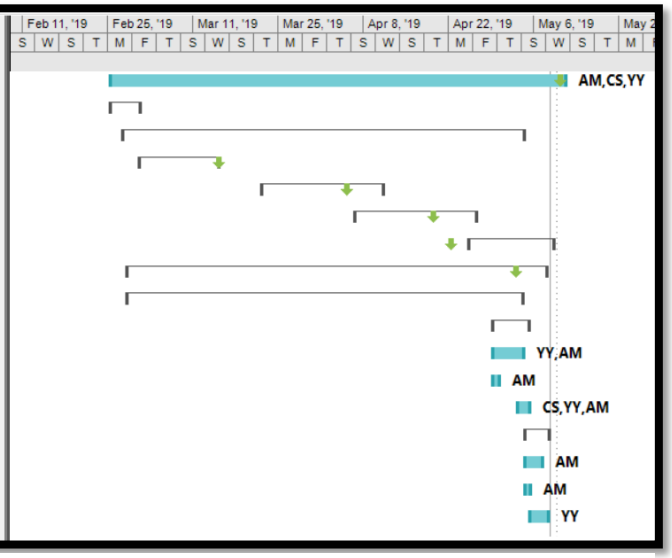


Figure C.7 - Deployment and documentation Gantt