UTS Faculty of Engineering and IT

48623 - MECHATRONICS 2

ASSIGNMENT 5

PROJECT REPORT

AUTUMN 2019

DUKE ST BOIZ
STOBOT – STORAGE ROBOT

DINH TUNG LE – 13302245
DAC DANG KHOA NGUYEN – 13302233
NGUYEN THANH TRUNG LE – 13301890
VICTOR OLUMIDE FAJEMIROKUN – 13301460
THANH TUNG VU - 13301902

Contents

I. Summary 2

II. Mechanical and Electrical Design 3

Mechanical Design (robot chassis) 3

Electrical design (circuit board) 4

Sensors 5

Actuators 5

III. Integration (ROS Network) 6

IV. Position Control 8

V. Localisation using Monte Carlo Localisation (MCL) 9

VI. Computer Vision and Machine Learning 11

OBJECT DETECTION 11

POSE ESTIMATION 15

FLOWCHART FOR ALL SECTIONS IN THE COMPUTER VISION PART 17

VII. Appendix 22

A. Source codes 22

B. Link to video 22

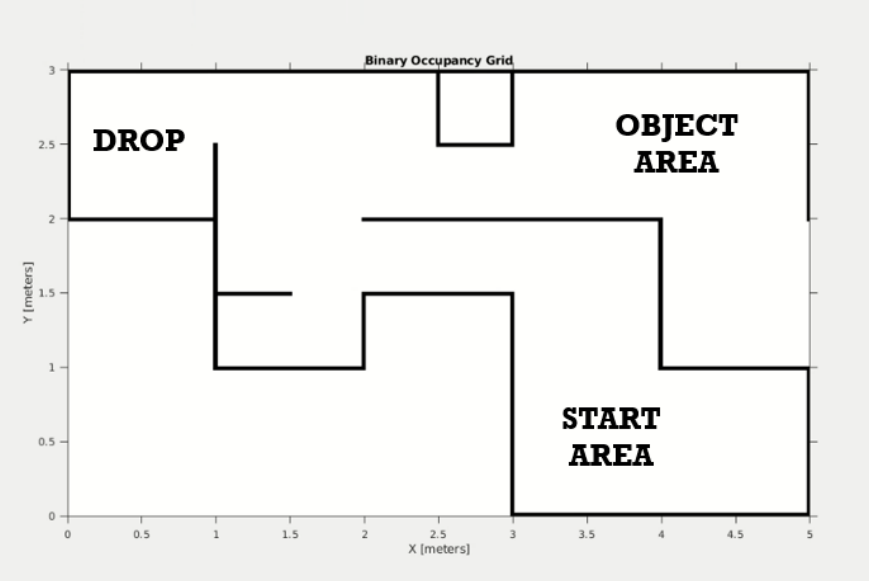
C. Member contribution 22

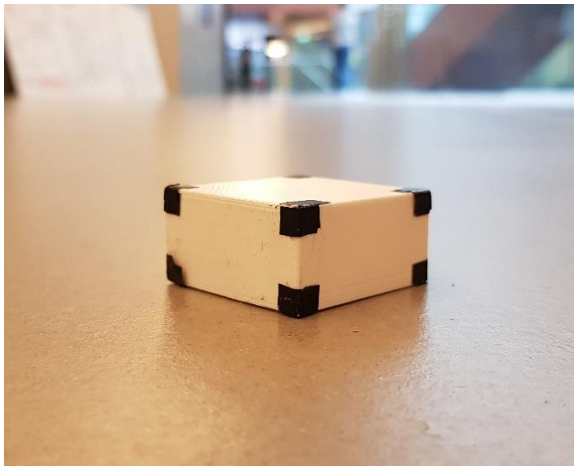
I. Summary

We are living in a world where automation is appearing in almost all aspects of our lives. We want to apply robotics into areas where robots can replace humans to do repetitive or risky tasks. One of which is storage management in a warehouse/industrial environment. For this Assignment, we decided to put together actuators, sensors, localisation and mapping, and computer vision in a robot that can move around on its own and manipulate objects automatically. We will achieve this by placing a Dobot, which is a 4-DOF robot arm, on a mobile base, and have it move real-world blocks, simulating what it's like to have such a robot in a warehouse.

The task of the robot is to:

- Find where it is within a maze
- Go to a region in the maze to find a known object
- Pick up the object and move it to a designated area



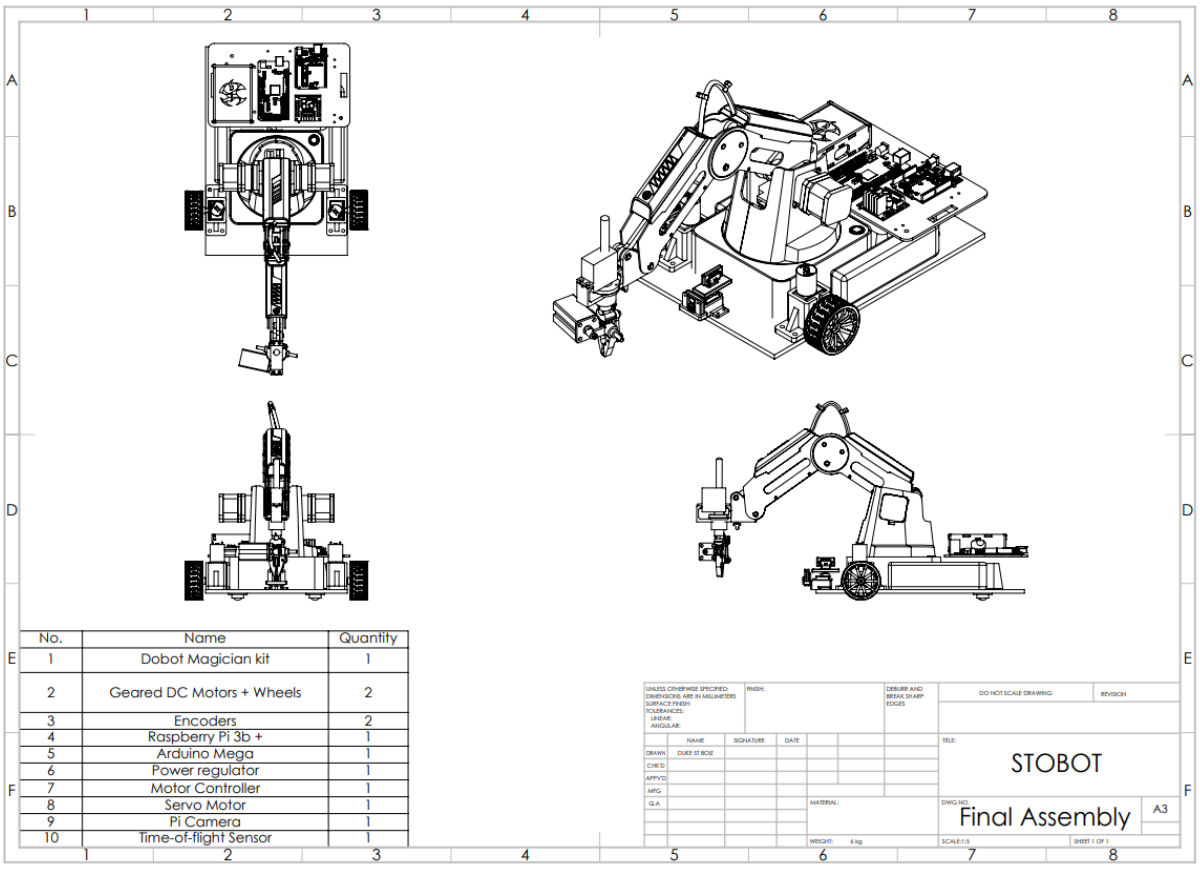


II. Mechanical and Electrical Design

Mechanical Design (robot chassis)

The robot is a Differential Drive robot, where the centre of mass (contributed the most by the Dobot) rests near the front wheels' axis. This assures that the centre of rotation is at the wheels' axis midpoint. There are 2 layers at the back, the lower one for the Li-Po batteries and the Dobot's pump. The upper layer holds the rest of the components.

Below is the assembly drawing of our robot. Made in SolidWorks.



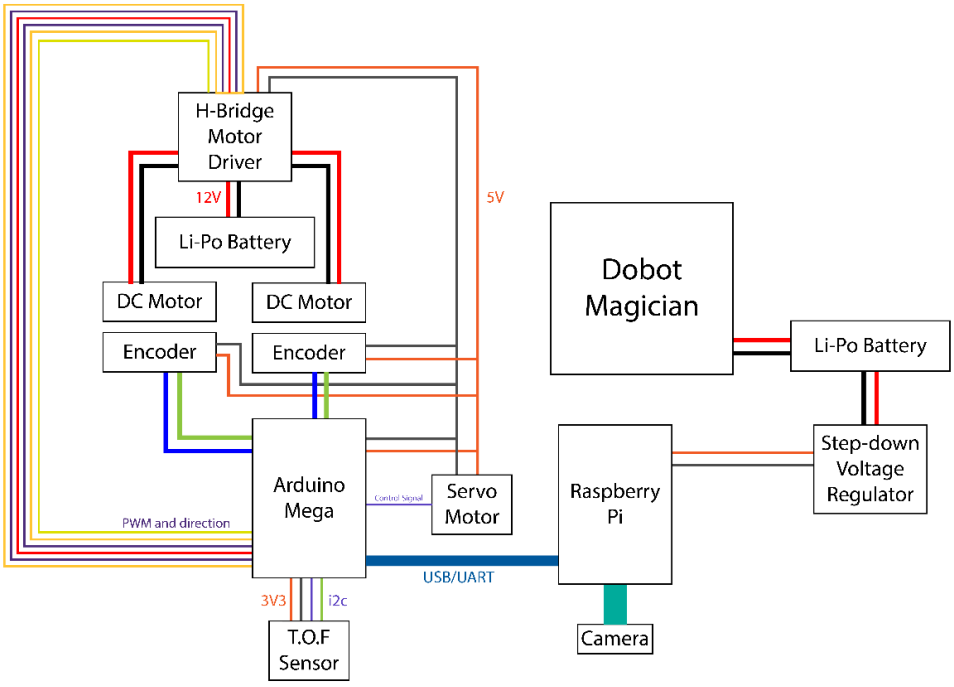
No.	Name	Quantity
1	Dobot Magician kit	1
2	Geared DC Motors + Wheels	2
3	Encoders	2
4	Raspberry Pi 3b +	1
5	Arduino Mega	1
6	Power regulator	1
7	Motor Controller	1
8	Servo Motor	1
9	Pi Camera	1
10	Time-of-flight Sensor	1

The components are:

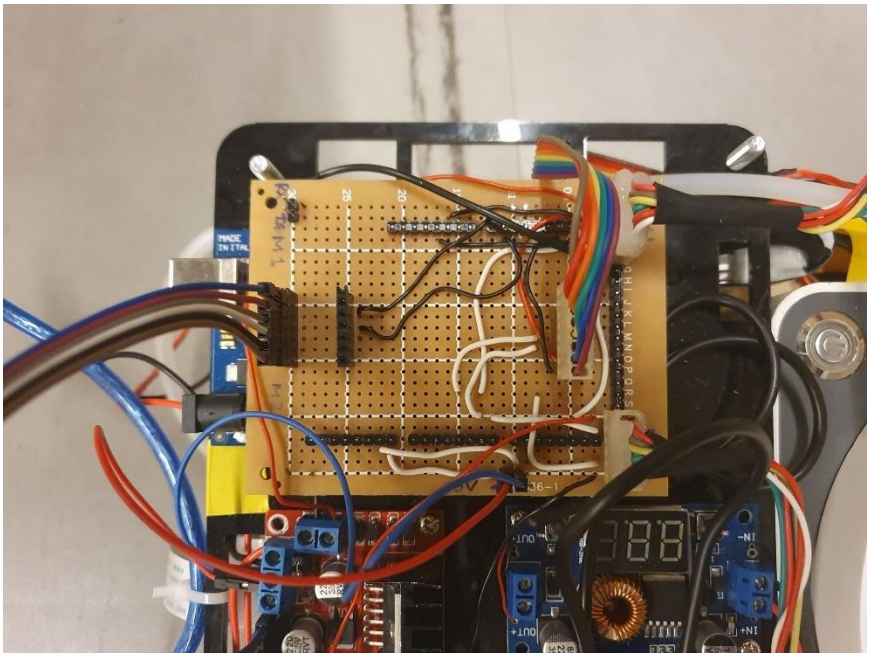
- Arduino Mega 2560
- Raspberry Pi Model 3 B+
- Dobot Magician robot arm
- 12V DC Motors (1:200 gear ratio)
- Time of Flight sensor VL53L0X
- Servo motor
- H-Bridge Motor Driver L298N
- Voltage Regulator

Electrical design (circuit board)

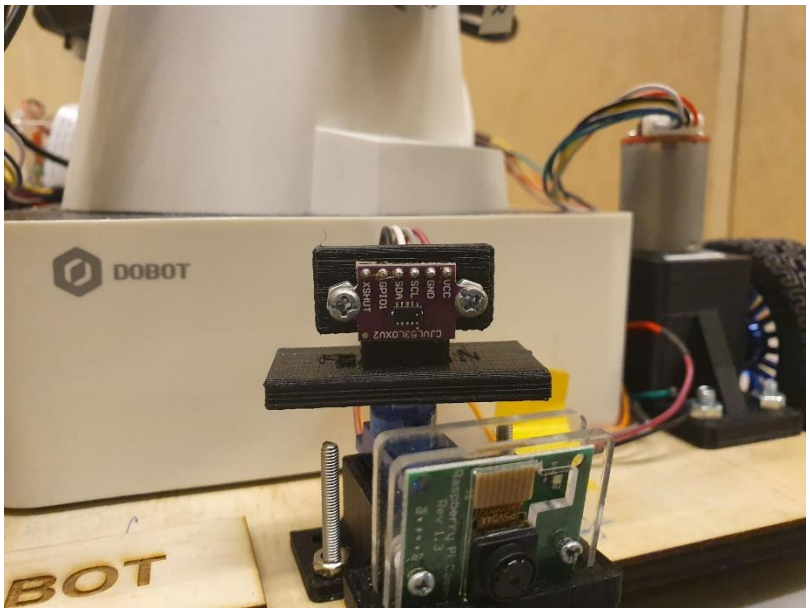
Below is the block diagram of the whole system, showcasing how they are all connected



For our Stobot wiring management, we discussed about the need of having a reliable electrical system (i.e. no breadboards and loose jumper wires). Therefore, we decided to design and solder our own circuit board that fits onto the top of the Arduino. The board took a lot of time to make. And we wished we had planned and simply ordered PCBs beforehand. However, the work paid off at the end, as we never have to worry about wiring ever again – everything is plug-and-play now.



Sensors



Being aware of the requirement of not using a Lidar for this assignment. We decided to construct our own, by mounting a Time of Flight (TOF) distance sensor on top of a servo motor and have it rotated around while the TOF records distances. A stepper motor would work better in this case (smaller angle increments), however we would need to find a way to "home" or find the current position of the stepper. We saved ourselves the trouble and went with the first idea instead.

Our servo rotates continuously from 0 to 180 degree and then back to 0 degrees, increasing 5 degrees each time. We tested with different kind of distance sensors, not just the TOF, including the IR from the second assignment. We chose to use the TOF because of its accuracy in short ranges (10-1000 mm)

The TOF sensor that we used was the GY-VL53L0XV2 Time-of-Flight Distance Sensor. According to the datasheet, different coloured objects affect the measurements differently. The colour of the walls of our Mx2 Maze is bright (very reflective), so it helped with sensing.

However, one major problem is the maximum range of the TOF sensor, which is 1.2m, much shorter than the maze's dimensions (up to 5m). Thus, in some conditions when the TOF sensor was pointed at a large empty area, it can't measure anything and automatically returns a value of 8191 – the out of range value.

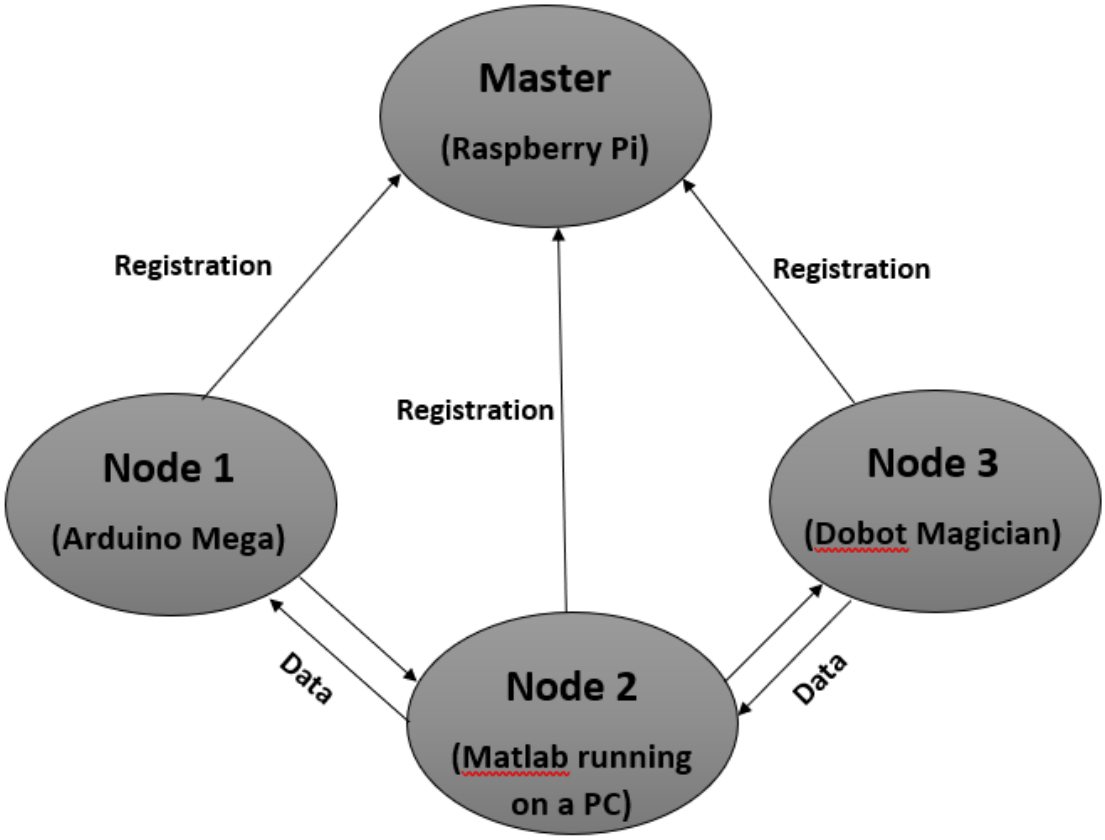
Actuators

We used 12V geared DC Motors with the gear ratio of 1:200 and the no-load speed at 12V of 30 RPM. The reason why we chose such slow motors was to increase accuracy in the movement of the robot; as well as providing constant torque as the weight of the robot is very heavy (~6kg).

The motors also came with Encoders, which helped us implement PID-control. At maximum speed, the rotation and step count of the Encoders are too fast for a normal I/O port of the Arduino to process. Hence, we had to connect the Encoders' pins to the pins which allowed external interrupts. Luckily, the Arduino Mega provided us enough interrupt-enabled pins to achieve this.

III. Integration (ROS Network)

Using all these different components, we had to setup a network to precisely gain control of the whole robot as one unit. We then setup a ROS network between all the components of the robot. The network looked something somewhat like the diagram below



Master (Raspberry Pi): The raspberry pi serves as the master for the ros network and all the nodes register to it in order to be listed on the ros network. The nodes communicate with the master to register subscribers, publishers and services. Once the nodes are registers on the network, they can now communicate with one another by sending and receiving data through topics and services.

Node 1 (Arduino Mega): The embedded system microcontroller, Arduino Mega is used in this project mainly for the control of the motors of the chassis and the servo, tof sensor combination serving as low budget lidar for the Monte Carlo Localisation. The mega is registered on to the ros network via rosserial_arduino to the master. The rosserial_arduino creates a serial communication between the Arduino mega and the raspberry pi and enables data to be shared via topics and services.

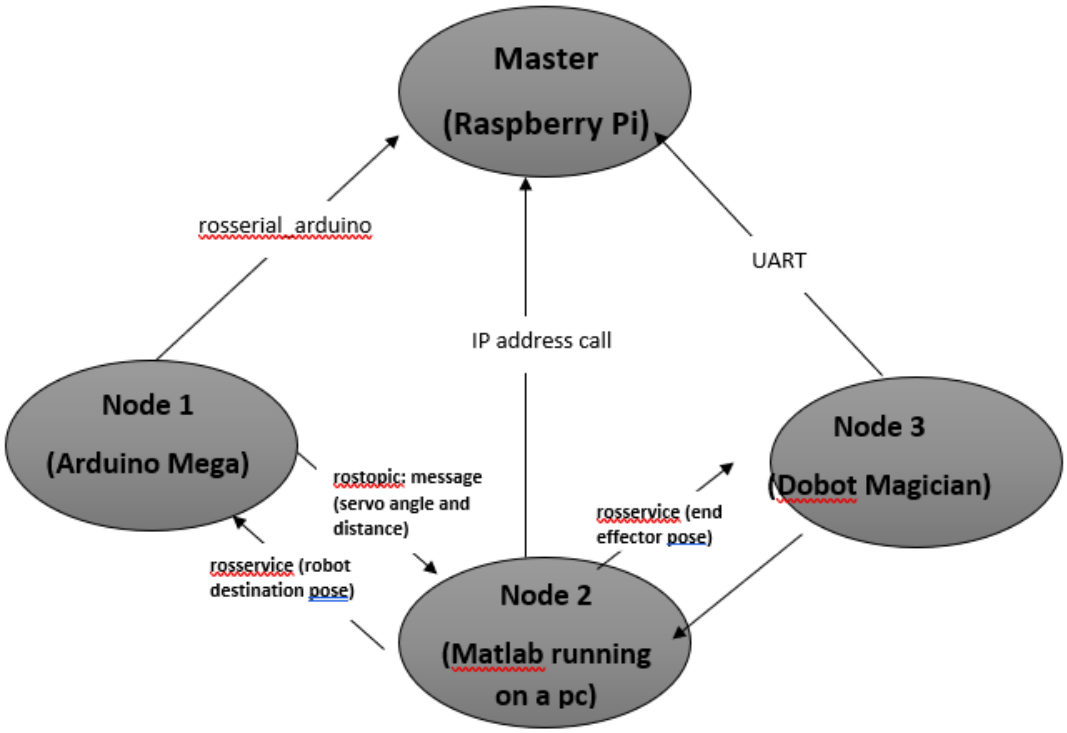
The data shared was between Node 1 and Node 2, the localisation performed on Node 2 sends the pose to a service which the Node 1 calls to and then reacts. Also, the lidar information published as a rostopic by the Arduino Mega and subscribed to by the Node 2.

Node 2 (Matlab running on a PC): This node connects to the master by addressing it by its IP address i.e. the IP address of the Raspberry Pi, the master. This node is the only node that shares data between itself and the other two nodes in the network.

Node 2 communicates with Node 1, sending out destination pose as a services and runs a subscriber that subscribes to a topic published by Node 1 containing the lidar information (servo angle, proximity reading) as the Node 2 performs the Monte Carlo Localization.

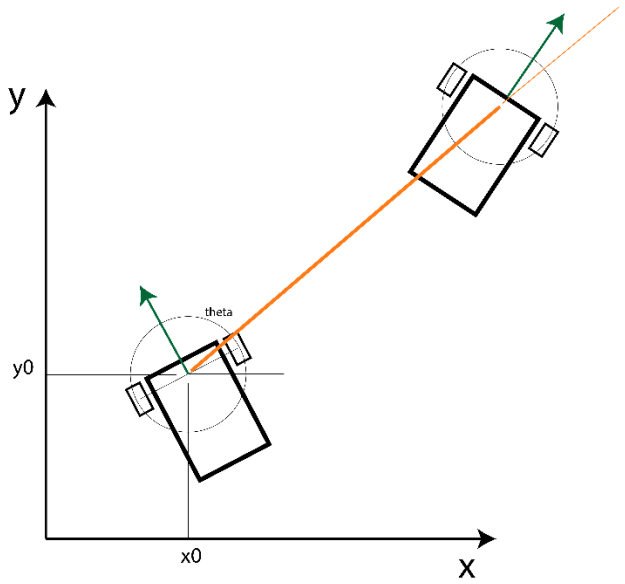
Node 2 also communicates with Node 3, sends out destination pose for the Dobot Magician to move to. This pose is obtained by the Computer Vision implemented using Matlab to detect the object and approximate the pose of the object.

Node 3 (Dobot Magician): Node 2 connects the ros network via a UART connection to the master, communicating with the master in order to call services. This node communicates with Node 2, receiving pose data for designated position of the end effector through a service.



IV. Position Control

Now that we had all the hardware stuff out of the way. We got straight into implementing software. Starting with having a coordinates-based control system where you can input a 2D Pose value (x, y, theta) in the maze the robot would go there accordingly.



The robot does this by storing its current position within the world frame. Then when it receives a destination point, it calculates the 3 moves to get there: one pure rotation, one straight path and one final rotation to reach the final Pose of x, y and theta. Once it has calculated the 2 angles of rotation and 1 distance of linear movement, it will translate these values into Encoder value setpoints, relative to the current Encoder values of each motor. Finally, it will try to reach these setpoints using a ***PID-Controller***, with parameters tuned to overcome the motors' natural damping characteristics (caused mainly by friction). The PID Controller also accounts for the error between the 2 wheels and corrects it by making the faster wheel slower through PWM; this ensures that the robot moves in a straight line when it is supposed to.

Once it has reached the target point, it will update its current position as the destination position, for future moves calculations.

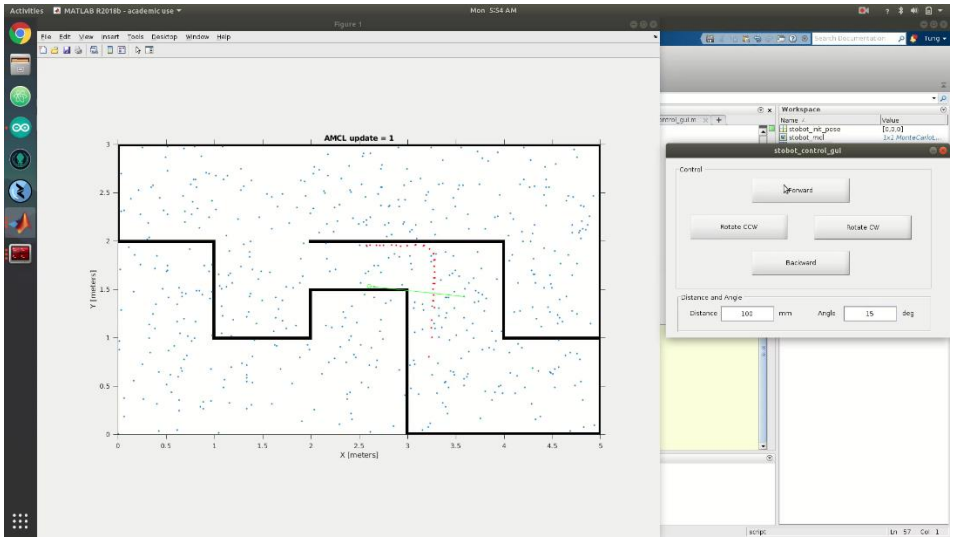
The programming structure is that each motor exists as a class, which has properties as current encoder value, setpoint and PID tuning parameters. Then there's a robot class that includes 2 of these motor "objects". This robot class would have properties such as current position, destination position and state (moving/stationary/calculating moves). And in order to make the flow of the program smooth and nonintrusive, we have written it such that the movement of the robot can run simultaneously with other parts of the codes (like the ones to turn the servo motor and gets distance measurements).

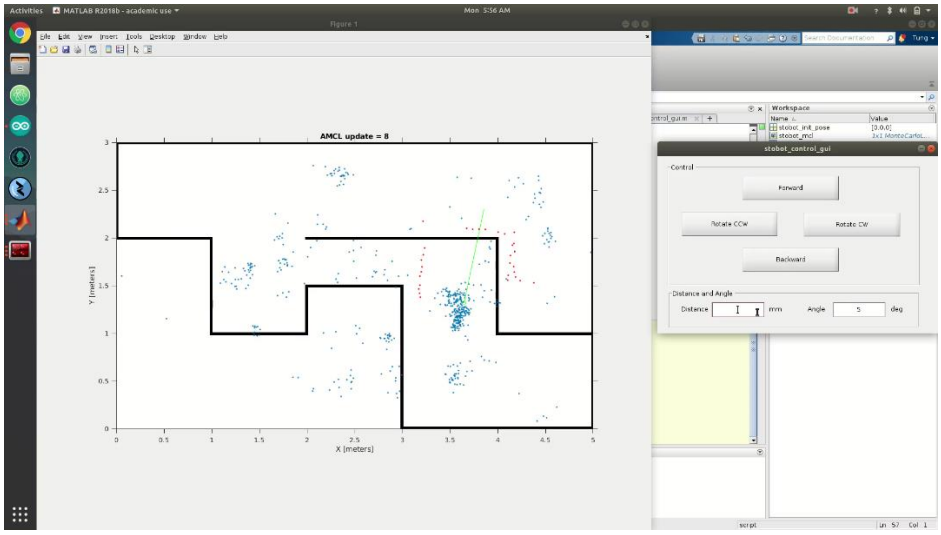
V. Localisation using Monte Carlo Localisation (MCL)

Although we know that our robot will start in around a certain area within the maze, we never know exactly where that is, and more importantly, the initial orientation of the robot. This is very crucial as the maze has long and narrow corridors, which if the robot enters with the wrong orientation or position, will crash into walls once it's in there.

Therefore, we need a way to determine accurately the robot's current position, then updates it every time the robot makes a move. One of the tutors of the subject suggested us the use of the Monte Carlo Localisation (MCL) Technique, which is a Particle Filter algorithm, used widely in robotics, especially probabilistic robotics.

MCL first generates all the possible poses of the robots, which at the start is all over the map. Then as the robot moves and measure distances to the environment that surrounds it, these points, or particles, are weighted based on their likelihood of being the correct pose. Then the particles are resampled, which means they are redistributed based on their weights. The process repeats until the points converge into a small enough area that we are confident that the robot is there. This "confidence" is called the covariance in the algorithm.



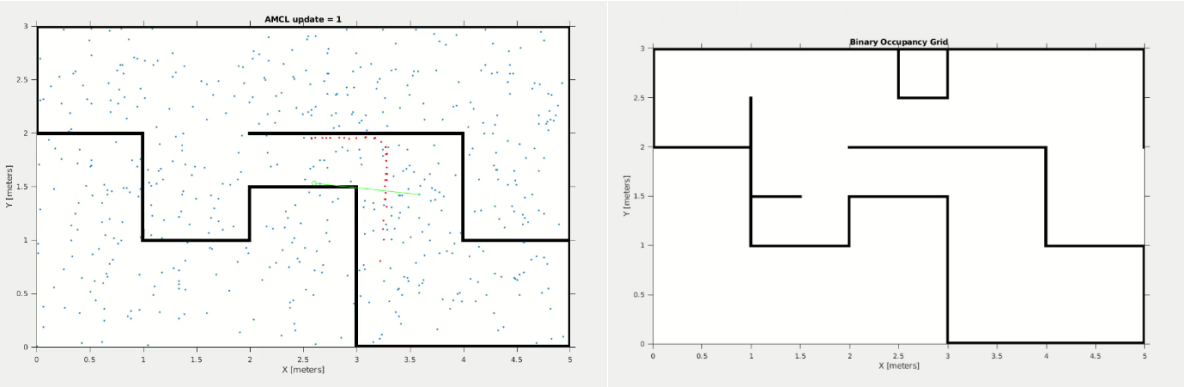


MCL takes 2 inputs: the robot's pose relative to the starting position (where it thinks it is) and a lidar measurement. We already have the pose from the Control section. What we needed was to turn our TOF readings into a lidar measurement. We were doing everything on MATLAB, so using MCL was straight forward. The most important aspect was to get the correct parameters, namely: the sensor's accuracy, the robot motion model and the way we resample and distribute particles.

To get the first initial values, we drove the robots around manually, then waited for it to take sensor readings, and repeated until the robot found its position. Once that was complete, the robot would move around the maze autonomously, through a pre-determined set of poses.

We faced many problems trying to get MCL to work, most of the problems arose from the fact that we did not have enough experience with how particle filters work, and what the parameters meant. But once we had known our ways around MCL, a real challenge presented: Our distance sensor's range was too limited, and the maze was too big. This made it hard for the algorithm to work with the default maze layout.

We fixed this by introducing our own layout, with more walls closer to the path of the robot, and adding more obstacles as well



The default layout is the one on the left and ours is on the right.

At the end, after a sleepless night of 2 of our members, MCL was working and robot moved autonomously.

VI. Computer Vision and Machine Learning

For the second main part of our project, pick up an object and bring it to particular location within the map, we decided to used computer vision and separated the task into four small sections:

- Detect object and move toward it
- Perform the object's pose estimation with respect to the camera's coordinate frame
- Perform a transformation between the camera frame and the Dobot's frame to find the pose of the object with respect to the Dobot's frame
- Have the Dobot pick it up and move back to the initial position.

Based on the above sections, the two main sections that play the most important role in the whole computer vision part. They are:

- **OBJECT DETECTION**
- **POSE ESTIMATION**

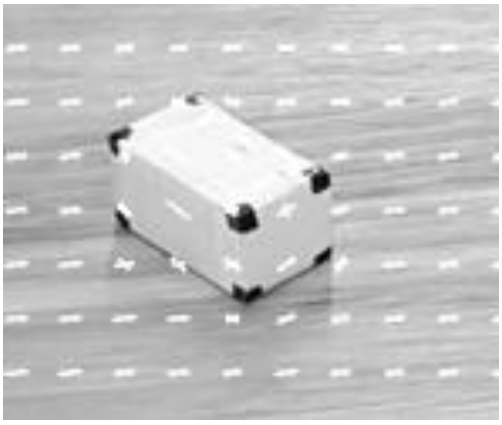
OBJECT DETECTION

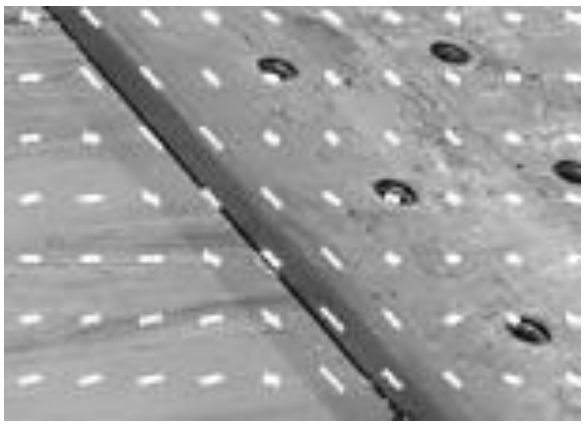
For object detection, there are multiple approach to this problem as it is a popular AI and robotics problem nowadays, and can be implemented by two main methods: using traditional Computer Vision – feature detectors, or deep learning – train an AI to detect a given object in a picture.

At the beginning, the method chosen was the traditional approach. There are also multiple methods for this scenario. One of them is using a Support Vector Machine Classifier (SVM) alongside with the features extraction algorithm Histogram of Oriented Gradients for object detection. Based on the benefits that this method brings, we have decided to use it.

1. SVM Classifier with HOG

The SVM approach was quite successful at the beginning, as the algorithm was able to detect the object from an image with a white background. However, it started to fail when the given image contains objects or scene that had a similar shape to the object (which contained straight lines or black dots). An example of the case where the classifier failed is given below. The metal rig had straight lines and black dots that forms the shape of a box. Because of that, the algorithm produced a high probability that the rig was the object that we were looking for.



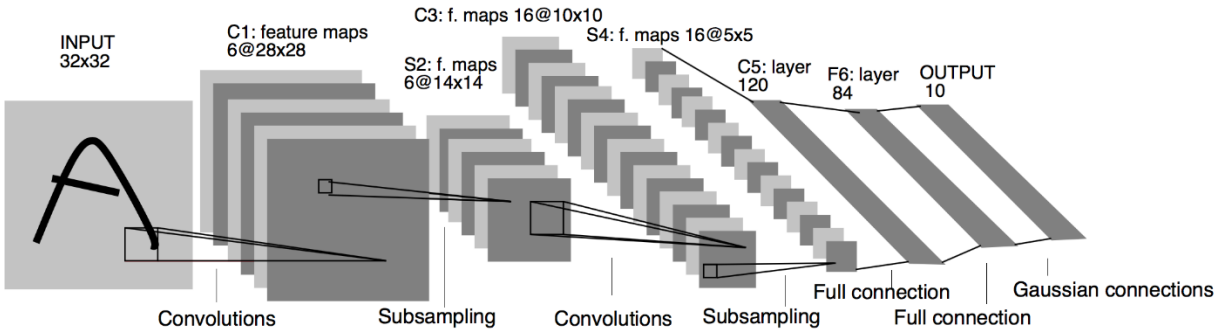


As the task is to locate the object, pick it up and bring it to a location, the first and also most important part – object detection – must be precise enough for us. We also tried increasing the number of the images in the dataset (originally 500 to 5000), but the results did not change much. There are two main reasons that were accounted for this:

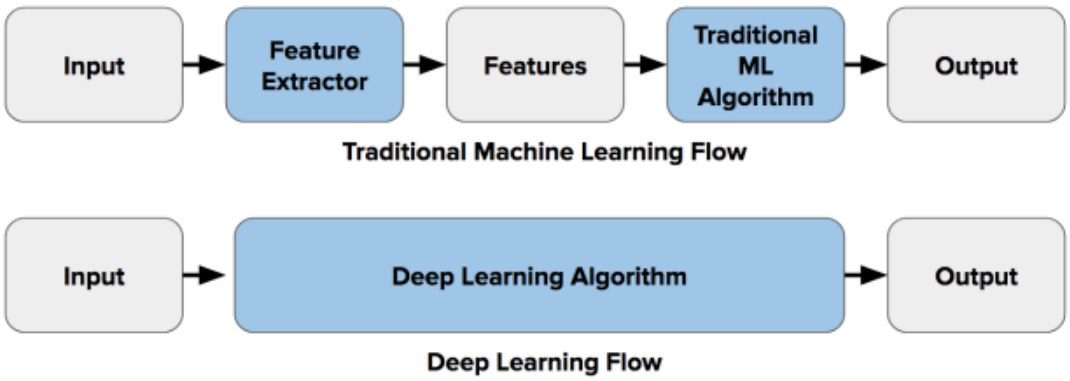
- The Histogram of Oriented Gradients is an algorithm used to extract the general features and overall shapes of an object, as the working principle of the algorithm relied on edges (gradients). Therefore, it would perform better if the given object has a bigger shape within the image and more distinct features (like vehicles, human faces). Our object was very simple and did not contain any special or distinct features, which made the data produced by the HOG algorithm unreliable.
- The SVM used the data produced from the HOG features, and as discussed above, the data was not reliable enough to use. Therefore, the algorithm did not perform well even with more given data.

2. Deep Learning

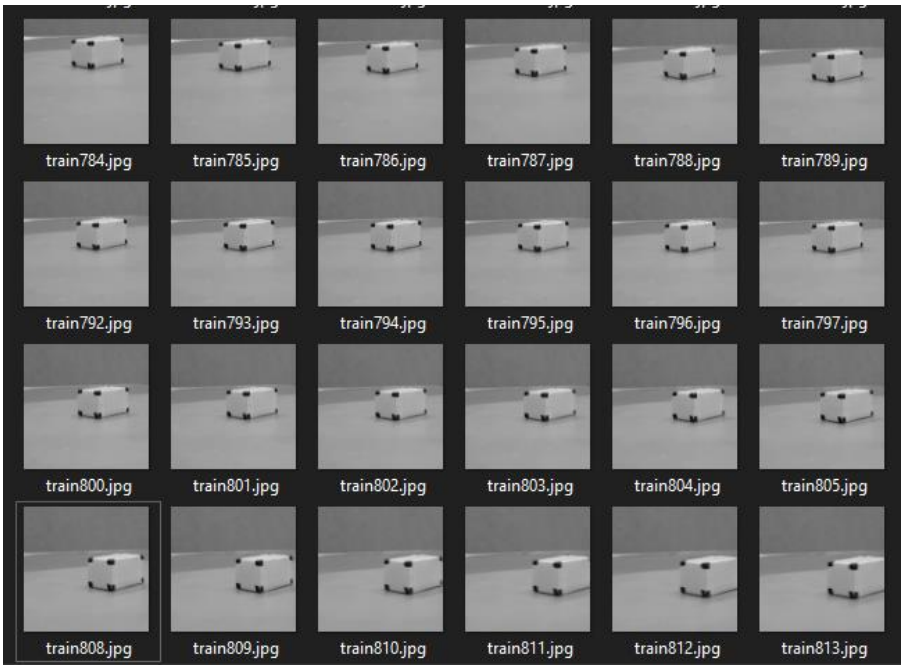
After some consideration, we have decided to use the second method – deep learning. A model was trained to find the object in the an image, and this model was created based on the structure of LeNet5, with some adjustment in the filter sizes and number of neurons in the fully connected layers, as LeNet uses a 32 x 32 image while our image was 128 x 128.



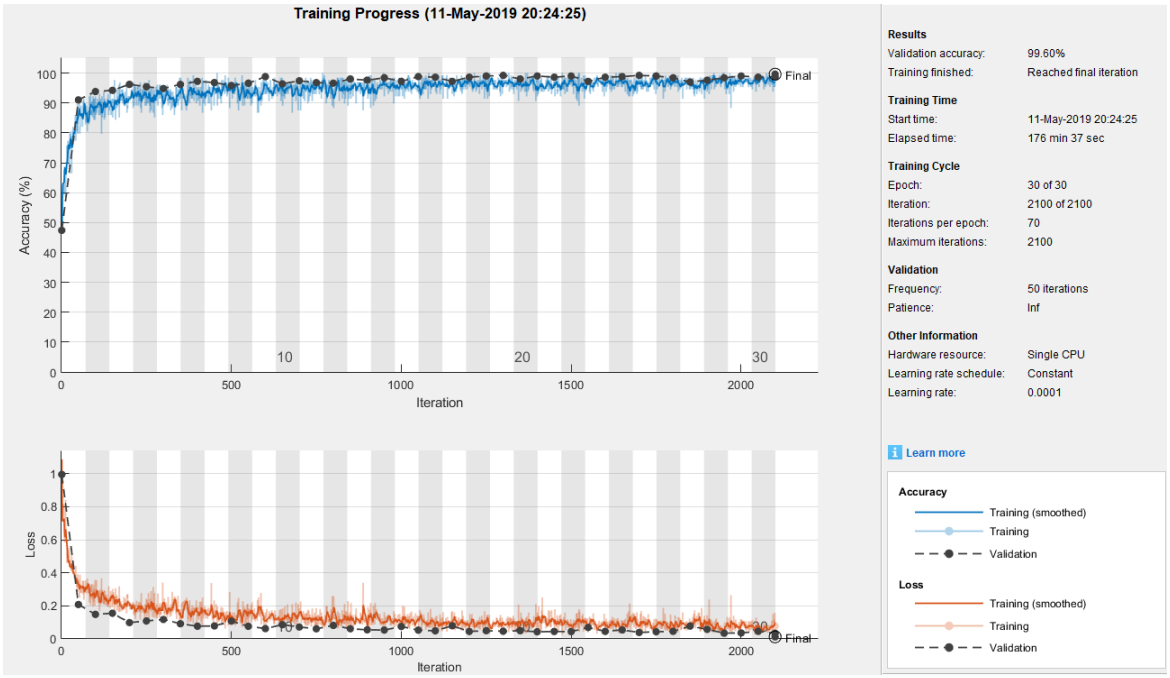
The deep learning method was much more straightforward than the traditional one. Basically we only needed to create more training dataset for the model as it would not be able to perform well with little amount of data. The picture below shows the differences in the implementation of the two methods.



The final number of images used for the training procedure of the model was 30,000. 10% of that number, which is 3000 images, were used for the cross-validation dataset.



The model took about 3 hours to train, with the final training error of 99.9 % and the cross-validation error of about 99.6%. We used the Adam optimizer (regularization included to avoid overfitting) method with the 30 epochs and a learning rate of 0.001. The outcome was quite satisfactory with the model now was able to detect the object in the cases where the SVM could not.



3. Discussion about the methods

The table below shows the comparison between the two Computer Vision methods for this particular project.

	SVM + HOG	Deep Learning
Advantages	• Training takes less time than deep learning • Can be combined with other features extractions	• Has high precision and accuracy • Produce the result instantly as it does not required any other features extraction algorithm in order to work.
Disadvantages	• Does not have high accuracy enough for this particular project • Results take longer time to be produced as it needs to go through a feature extraction algorithm first before the classifier.	• Training takes a lot of time

The main reason for the better performance of the deep learning model was that it extracted the features directly from the image by applying filters through it multiple time. By doing so, some of the key features of the object such as the black corners were kept and used for comparison. Therefore, it was able to distinguish between the object that it needs to find and other objects in the image.

The object chosen was designed with white body and black corner intentionally. As the task was to use to Dobot to pick it up, the object was expected to have features that can be easily extracted just by using a feature extractor or two, as we do not have good and enough knowledge about computer vision and features extraction to perform such tasks. That is why the object has a very simple design, so that the black corners can be detected easily. However, because of the simple look of the object, it makes the detection task difficult as there is no distinct or special features about the object. It requires a much more powerful method in order to get the task done, but the method of deep learning is feasible enough for us to implement as there are a lot of instructions and tutorials online. In conclusion, for this particular, it is a trade-off scenario between the features extraction problem and object detection, as object with multiple features will make the detection part easier but will also create more difficult tasks for the features extraction, while on the other hand, a much more simple object with less features will make the object detection more difficult but the features extraction will be much more straight forward.

Our group chose the second approach as the features extraction is harder to be implemented than the object detection. This topic will be further supported in the pose estimation section.

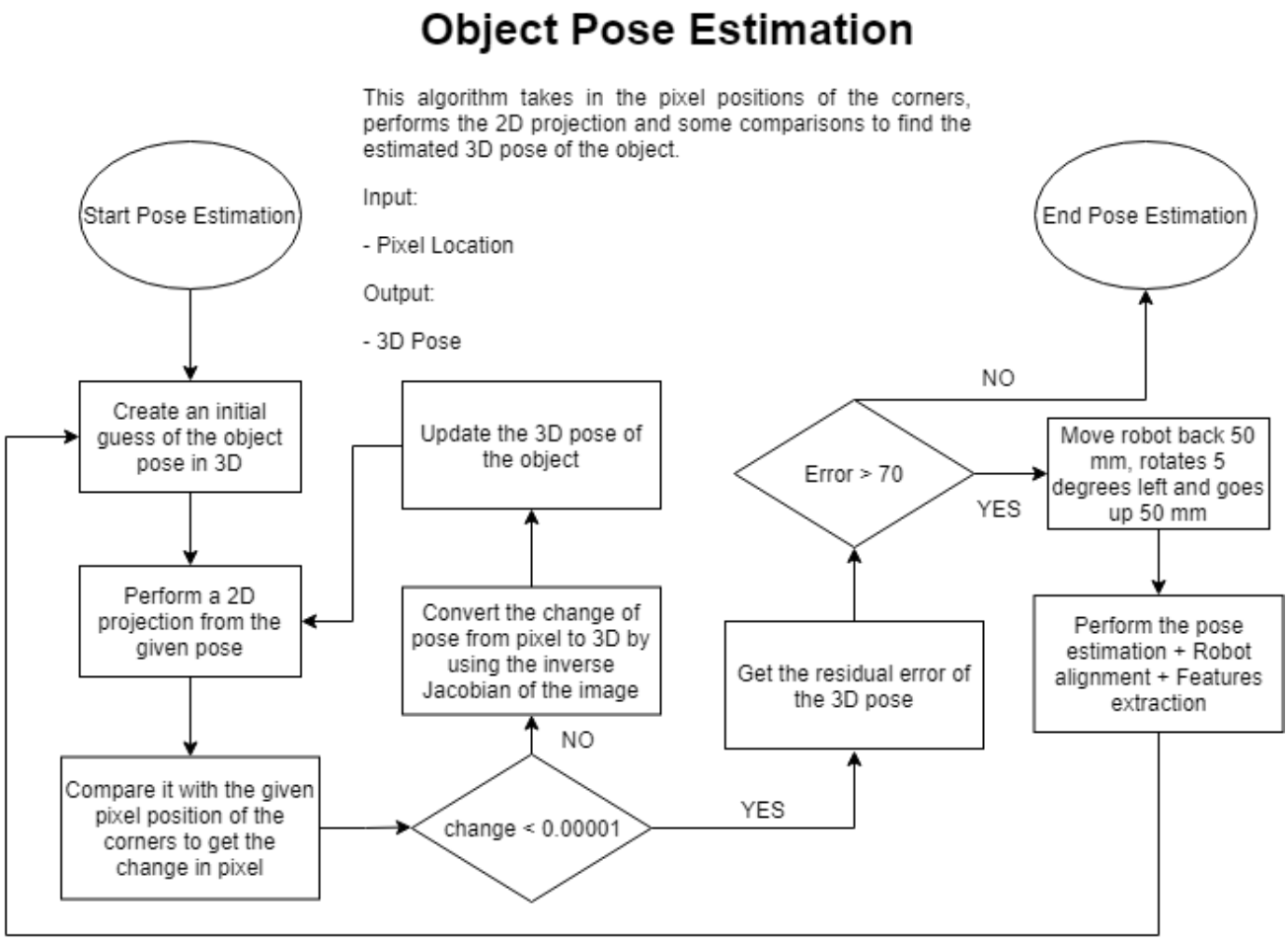
POSE ESTIMATION

For pose estimation, it is required to know at least six distinct features from the object. The reason is in order to represent an object in 3D, we need 6 axes, which are 3 for translations and 3 for rotations. Furthermore, it is also crucial to know the exact position between each individual feature in the object coordinate frame.

Due to the above reasons, the object should contain simple features that can be easily detected using one or two features extractor algorithm. If the object is too complicated, then by using the features extractor, it would be very hard to get the distinct features of the object. Even if the features are detected, it is impossible to tell the distance between those features in the object coordinate frame. This is the trade-off problem that was mentioned in the object detection section above. It is required to sacrifice the feasibility in the object detection part to get a much more implementable pose estimation part.

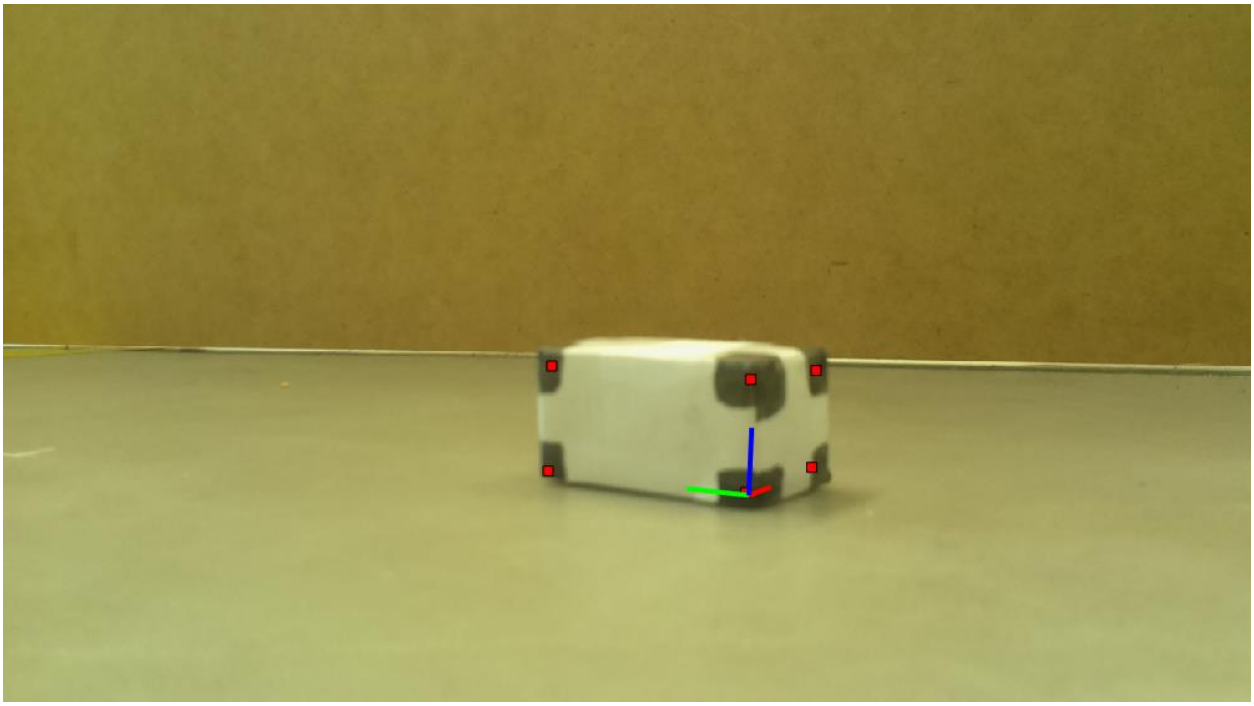
1. Method

The method of the pose estimation can be described using the following flow chart:



By comparing the projection of the object pose from 3D to 2D with the features extracted from the image and minimizing the error, we will be able to obtain the pose of the object eventually.

One of the must crucial part of the pose estimation was to calibrate the camera properly and get the camera matrices. From the camera matrices obtained from the calibration, the 3D to 2D projection, which requires that matrix, was able to be performed.



The image shows the final result of the pose estimation algorithm, where the red dots that represents the estimated pose of the object in 3D align with the black corners on the object. From the given pose, an illustration of the coordinate frame is drawn to show the frame of the object.

2. Discussion about the method and result obtained

The final result from the pose estimation algorithm was not as precise as expected. For the horizontal and vertical element, which stand for the x and y axis of the camera frame, respectively, the result was accurate and ready to be used with the Dobot cartesian control. However, the essential factor, the depth element, was not so reliable.

Luckily, we found that the inaccuracy was just an offset that was proportional to the value of depth obtained from the algorithm. From the data collected using the algorithm with real measurement, the best fit line was constructed that described successfully the offset in relation to the depth value.

Another issue was the orientation of the object. As the depth value of the estimated pose was not correct, it affected the orientation of the object with respect to the y axis of the camera as well (the value that we care the most). Therefore, to account for that inaccuracy, we decided to change from the gripper to the suction cup. The suction cup did not require the full pose of the object, unlike the gripper. With the suction cup, we were able to expand the surface area of the object to increase the chance for it to be captured by the Dobot.

3. What we would have done differently

There are many options for us to try in future projects. They are listed below:

- Stereo Vision
- RGB-D Camera
- Eye-on-hand robot configuration

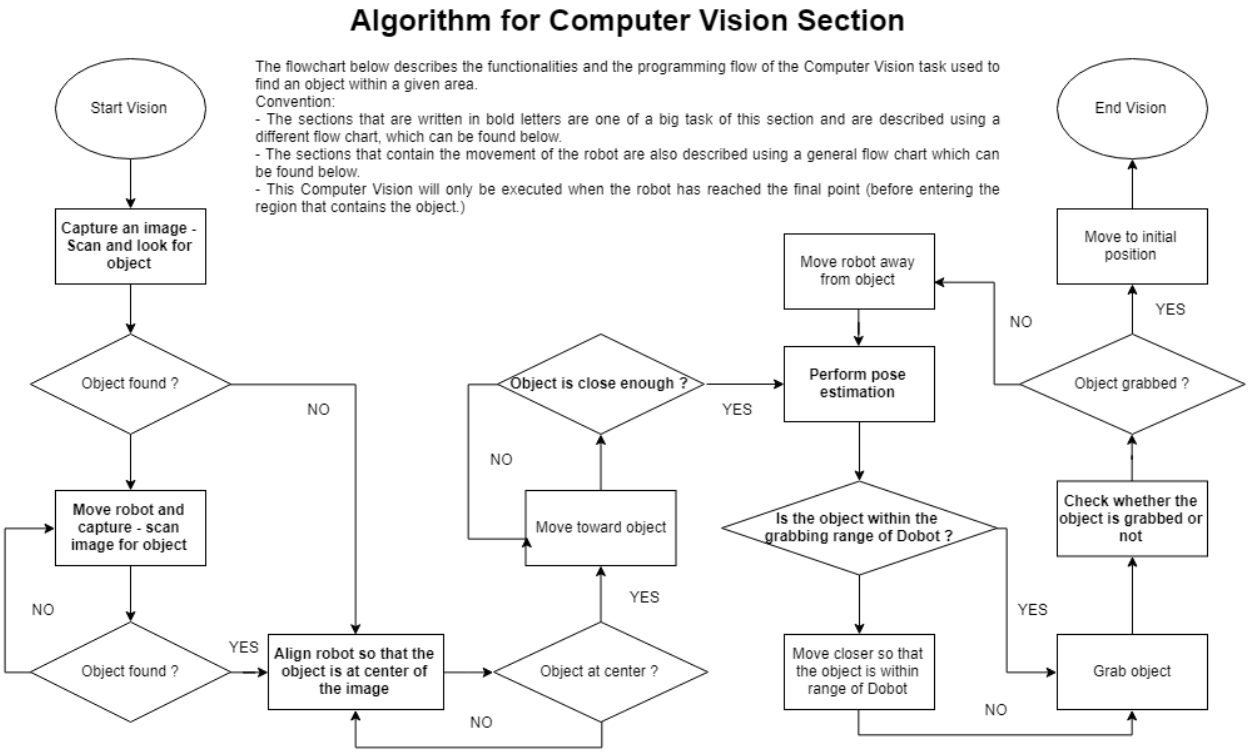
For Stereo Vision, it will allow us to have a better pose estimation result, as the fused signal of the 2 cameras gives us 3D point cloud. The drawback of this method is that it is quite difficult to be implemented in such short amount of time.

For RGB-D camera, it is the same as Stereo Vision, but more robust and reliable. It will also allow to use the data from the camera to perform and get better localization result.

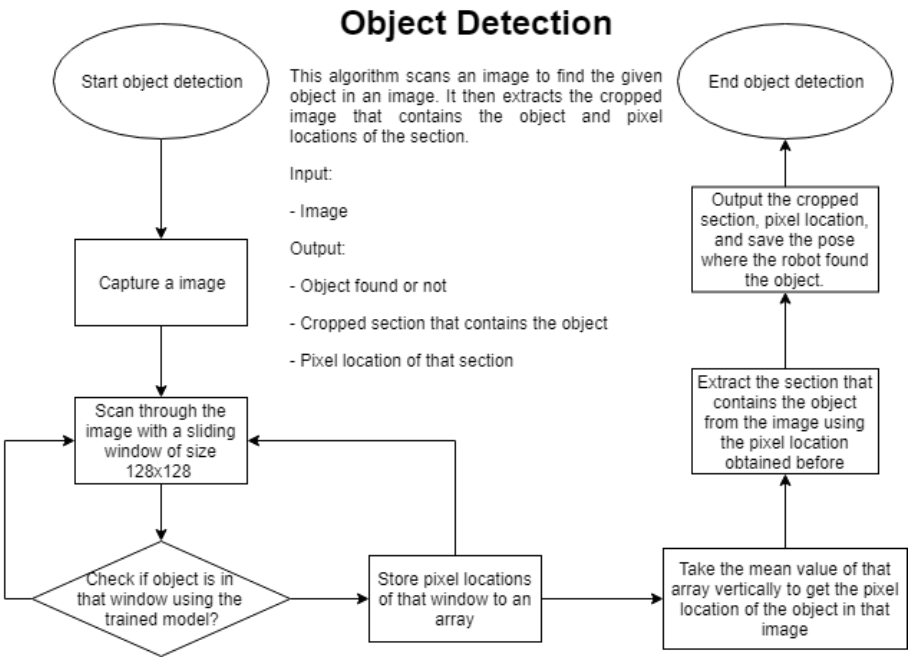
For Eye on hand, it will give us a better control over the Dobot arm rather than relying on the stationary camera at the front of the robot. However, it will be much more difficult to get a precise value of the camera pose with respect to the Dobot frame, as the camera moves with the end effector.

FLOWCHART FOR ALL SECTIONS IN THE COMPUTER VISION PART

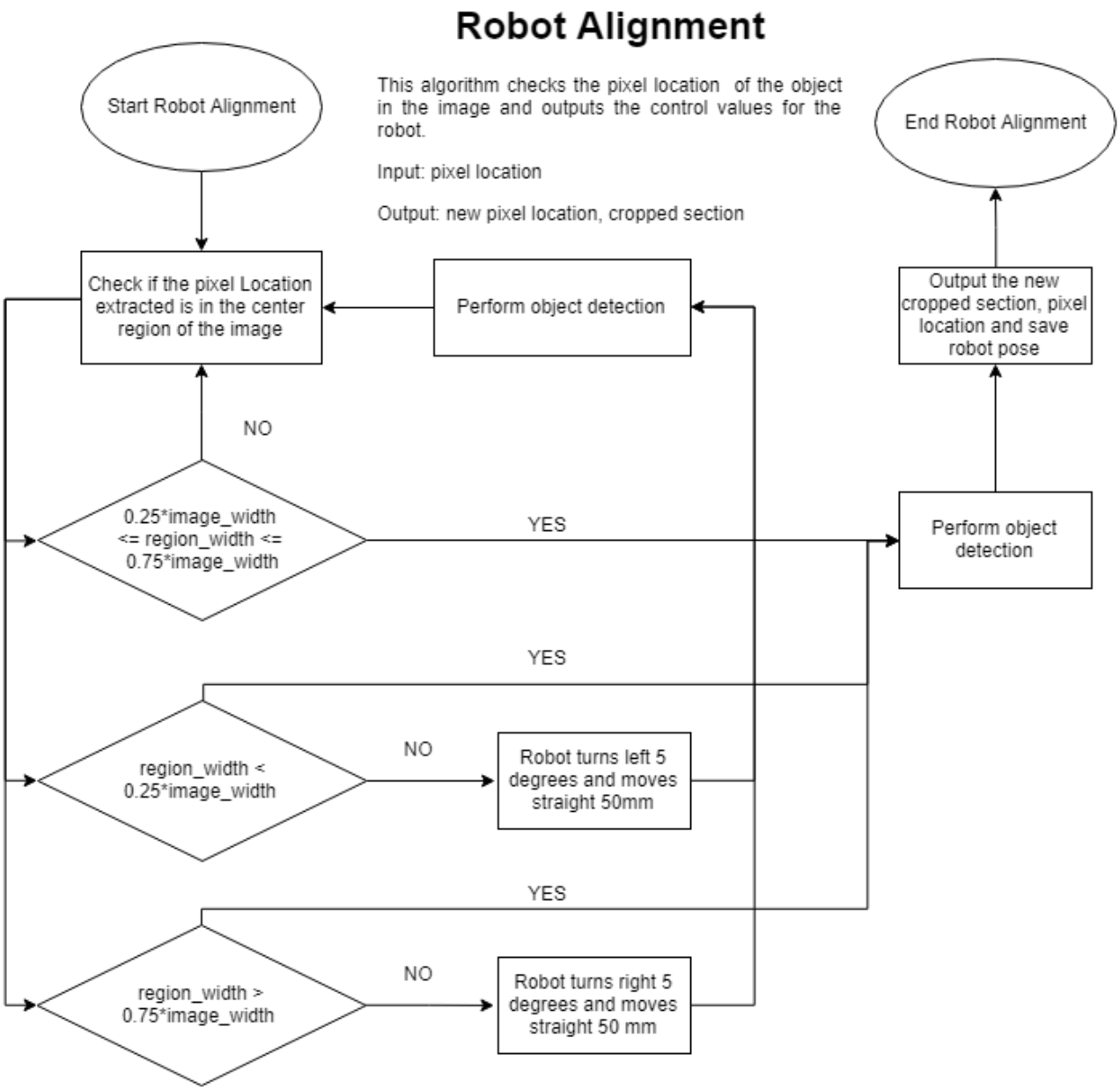
Main Programing flow



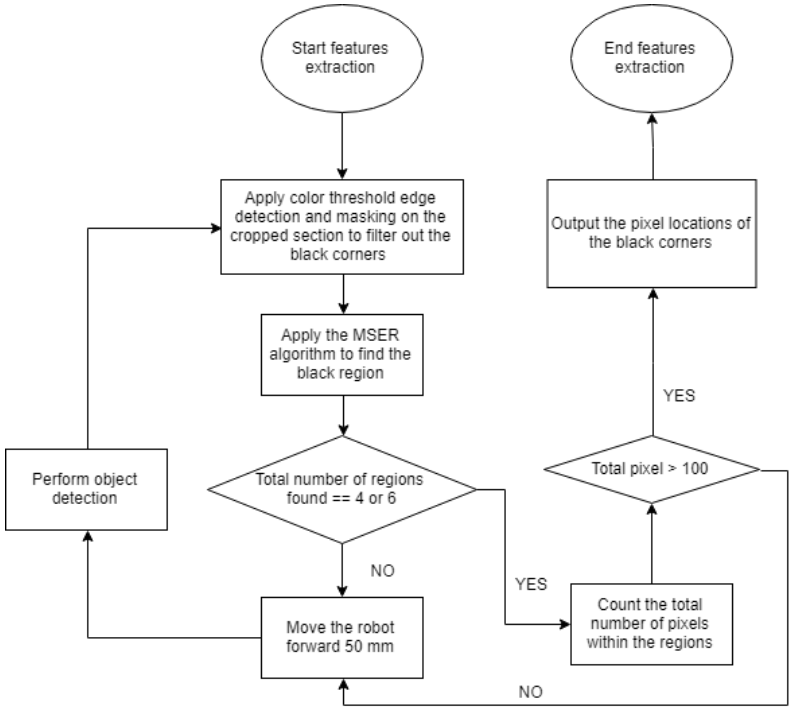
Object Detection



Robot Alignment



Feature Extraction



Features Extraction and Distance minimize Algorithm

This algorithm takes the cropped section that has the object and performs the features extraction algorithm MSER to find the pixel location of the black corners of the object. It then checks the total pixel number of the regions and moves closer in case the number is smaller than a threshold.

Input:

- Cropped image that has the object

Output:

- Pixel Locations of the black corners

Object Pose Estimation

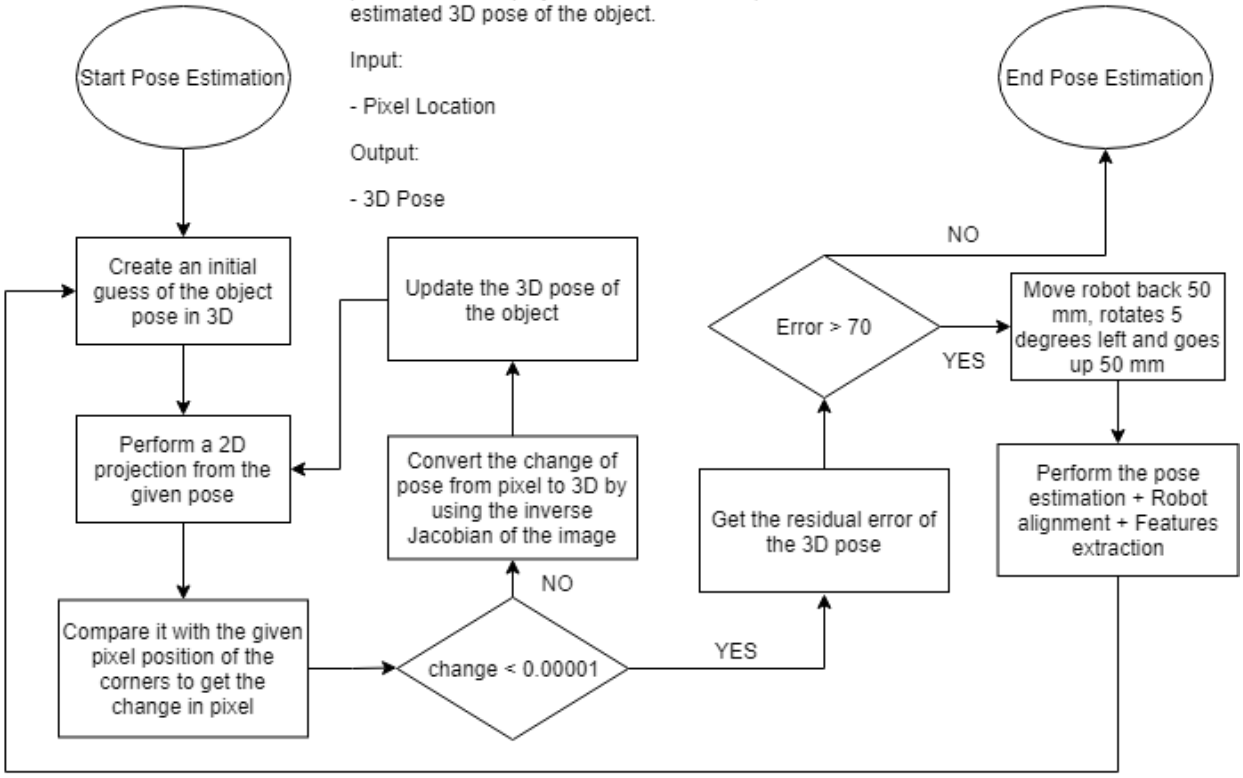
This algorithm takes in the pixel positions of the corners, performs the 2D projection and some comparisons to find the estimated 3D pose of the object.

Input:

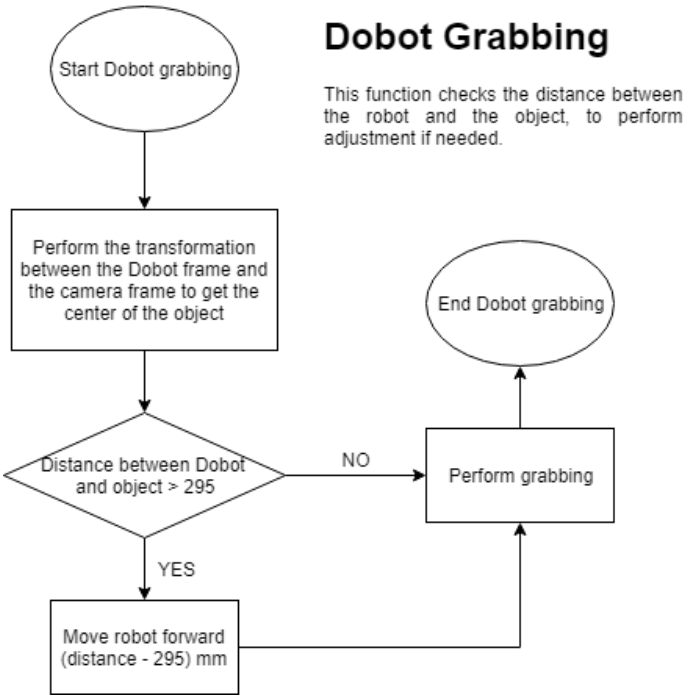
- Pixel Location

Output:

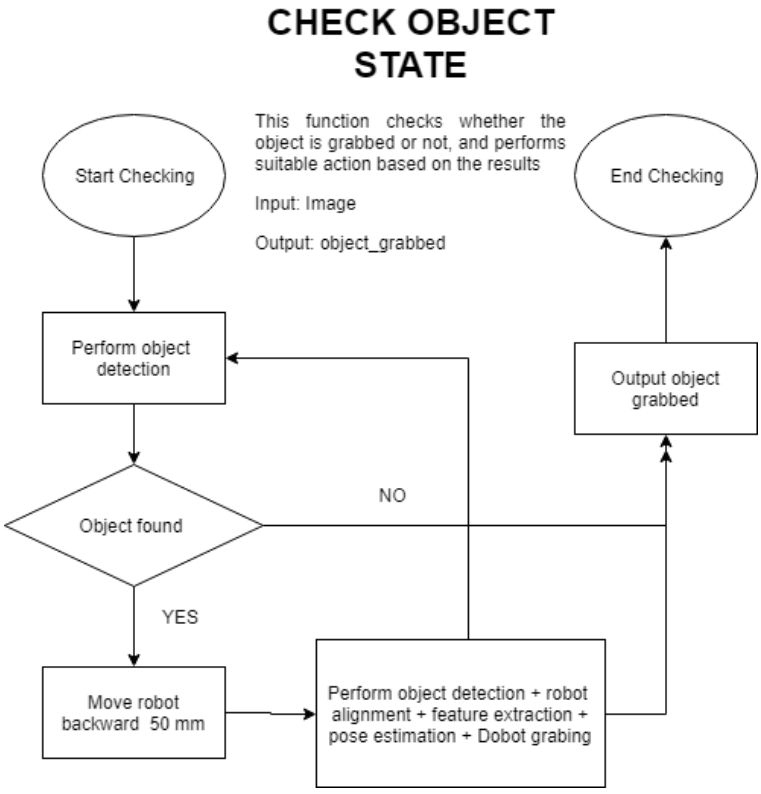
- 3D Pose



Dobot Grabbing



Check if the object has been grabbed



VII. Appendix

A. Source codes

http://tiny.cc/StobotSource

B. Link to video

http://tiny.cc/StobotVideo

C. Member contribution

Group Member	Contribution	Task
Dinh Tung Le	30%	- Electrical Design - Position Control - Localisation
Dac Dang Khoa Nguyen	30%	- Mechanical Design - Computer Vision - Dobot Control
Victor Olumide Fajemirokun	15%	- ROS Network - Arduino – Pi integration
Nguyen Thanh Trung Le	15%	- Sensors and Encoders - Components testing
Thanh Tung Vu	10%	- Helping with researching on Localisation - Helping with building the robot