# Embedded Visual Control 5LIA0 Assignment 2 – Group 3

| Author | Student ID | Email |
|---|---|---|
| Sergio Gerónimo Parras | 2048396 | `s.geronimo.parras@student.tue.nl` |
| Shashank Venkatesan | 2022036 | `s.venkatesan@student.tue.nl` |
| Tomas van den Heijkant Bataller | 1836080 | `t.v.d.heijkant.bataller@student.tue.nl` |
| Varun Kumar | 2022559 | `v.k.siva@student.tue.nl` |
| Vasilis Rizeakos | 2036908 | `v.rizeakos@student.tue.nl` |

June, 2024

## 1 High-level description of the proposed function

Using various concepts of robotic motion control and visual control learned in early stages of the course, we design a simple project that is able to demonstrate and improve on them. The main goal of the project involves doing certain 'tricks' with the Duckiebot. This essentially involves the Duckiebot performing a set of manoeuvres around the shoes of a person.

- Recognize specific shoes in front of the camera.
- Distinguish and classify different shoes.
- Estimate its own position on a virtual map.
- Track the current position of shoes on a virtual map.
- Perform specific-to-each-shoe maneuvers.
- Plan and follow paths based on the open environment.

## 2 Requirements

Based on the high-level description of the system, the following requirements are extracted:

| ID | Description | Success Criteria | Priority | Remark |
|---|---|---|---|---|
| R1 | The system shall be able to identify April tags and estimate the position of the duckiebot in the local environment | Highly accurate estimation of the pose of the duckiebot should be obtained | mandatory | The id of the local tag and its related data can be read |
| R2 | The system should be able to recognize shoes by using the camera | Shoes are correctly marked by a bounding box most of the time | mandatory | 90% images with shoes are correctly marked |
| R3 | The system should be able to classify different colored shoes | Different colored shoes are given different IDs most of the time | mandatory | Classifications with high confidence and low false positives should be given |
| R4 | The robot should be able to keep track of its position via odometry. | Only 2 cm deviation after performing a 1 by 1 meter square maneuver | mandatory | A combination of PID and Kalman filters to estimate position and orientation |

**Table 1 continued from previous page**

| | | | | |
|---|---|---|---|---|
| R5 | The robot should avoid obstacles | It does not bump into obstacles | optional | Use a combination of distance estimation and Time of flight sensor |
| R6 | Estimate the distance to shoe based on the image and size of bounding box | Obtain distance with a margin of 10 cm from the correctly classified shoes. | mandatory | Use image rectification technique and trigonometry to estimate distance |
| R7 | Estimate the global position of a shoe given the distance and triangulation | Obtain the shoe position with a margin of a radius of 20 cm | optional | April tags |
| R8 | Detect distance between two tracked shoes | Obtain the distance between two shoes given the estimated position | optional | Camera to estimate distance from shoe and find it's pose |
| R9 | Perform a maneuver given that the distance between two shoes is greater than a certain value | The maneuver is only done if the robot successfully detects a distance greater than a certain value | optional | Generate paths using trigonometric functions |
| R10 | While performing the manoeuvre, the duckiebot should not deviate from it's planned path | The duckiebot is able to successfully able to perform the 'tricks' without any collision or steering of the intended path | mandatory | Use PID to steer the vehicle on course with feedback from encoders |

Table 1: Project Requirements

## 3 Proposed approach

### 3.1 General approach

The approach is separated into three different parts namely the visual control, motion control and path planning with the visual control forming the chunk of the operations. A Duckiebot uses two CNN's to carry out its tasks. One CNN responsible for detection detects shoes and people in the camera's frame of view and creates bounding boxes around the detected objects. The resulting bounding boxes are then forwarded to the next CNN that classifies the bounding box among five different types of shoes. The camera also plays two other important functions of estimating distance from the detected shoe and detecting April tags for localization and deriving the pose of the Duckiebot in the environment.

After a particular type of shoe has been identified in the frame of view, the planning part of the application gives instructions and paths for the Duckiebot to follow. The motion control helps the Duckiebot to stay on track using error correcting feedback loops.

### 3.2 Object detection

For the robot to move in the real world, it needs timely updates of its environment. While, it can roughly calculate its position based on the wheel encoders, it also requires feedback on the place of objects and obstacles on the way. This can be done using light sensors, such as Time of Flight sensors, but also with camera feed for more accuracy and information. In our implementation, we subscribe to the camera node of the Duckiebot, which provides a distorted view of the world (due to the fish-eye camera), and use the Yolov5 Object Detection model to identify the existence of people and shoes in the frame. Yolov5 will provide classification among these two labels as well as the bounding boxes of the corresponding object in the distorted image. The bounding box contains the (x,y) coordinates of the top left and bottom right corners of the box. As we are also interested in the distance and angle of the object with relation to the robot, we need to perform Visual Distance Estimation as well, as described in Section 3.4.2. The two results are stored in a message of type Rect along with the bounding boxes of each shoe detected by the neural network. The information is encoded along with a CompressedImage message type of the distorted image captured from the camera into a SceneSegments message type. This message is relayed to the Shoe Classification Node which is described in Section 3.3. Figure 1 depicts the overlay of the undistorted camera feed with the bounding boxes of the identified objects and their orientation with respect to the robot.

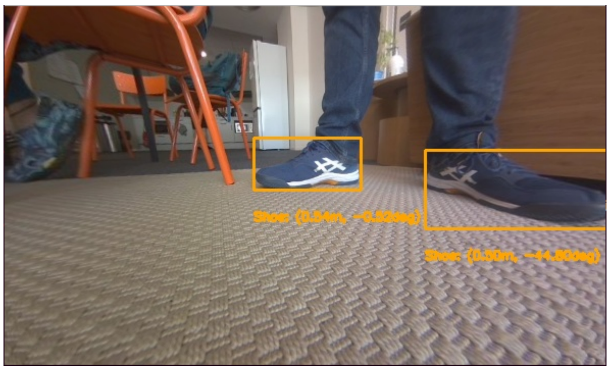


Figure 1: Debug window of the Object Detection node. The image taken from the camera feed is undistorted and the bounding boxes of the classified objects (people or shoes) are overlayed along with the distance and angle of the Duckiebot w.r.t. the object.

### 3.3 Image classification

Since our application has different behavior depending on the shoe it detects while also estimating the position of each shoe found, we require an intermediate node responsible for classifying the shoe object to one of our five labels. The Shoe Classification node subscribes to the topic published by the Object Detection node, which contains the original distorted image, the identified bounding boxes along with the distance and angle calculated in Section 3.4.2 and then classifies each shoe to either "Sergio", "Shashank", "Tom", "Varun", or "Vasilis". Then, it places the distance and angle measurements received to a position of a 10-object array depending on the classified label. We create an array twice the size of our number of classes to account for both feet of each person. In the array, each position contains the position, angle and a dirty bit to specify whether or not the current shoe was classified or not. If is does, it updates the position values and clears the dirty bit. Otherwise, it retains its previous values. Finally, it publishes the array as a PointCloud message, which is then used by the Pose Estimation node.

The Image Classifier was implemented using a CNN model trained with images sampled from the camera feed after being detected by the Object Detection node. Although each shoe was chosen to be different from the others, the CNN was more confident in the classification of shoes with more vibrant colors, such as blue and green. In the other cases, the shoes are sometimes misclassified because of their secondary colors, i.e. brown, white and grey. Therefore the CNN was trained with a large dataset of more than 5000 images to minimize such errors

### 3.4 Object pose estimation

The robot needs to be able to know what location occupy objects in its surrounding, so as to build a virtual map and plan its next moves. Since we have no made-for-this-purpose tags or any element that is easily trackable, we need to make use of a combination of techniques to track the desired objects.

#### 3.4.1 Time-of-Flight sensor

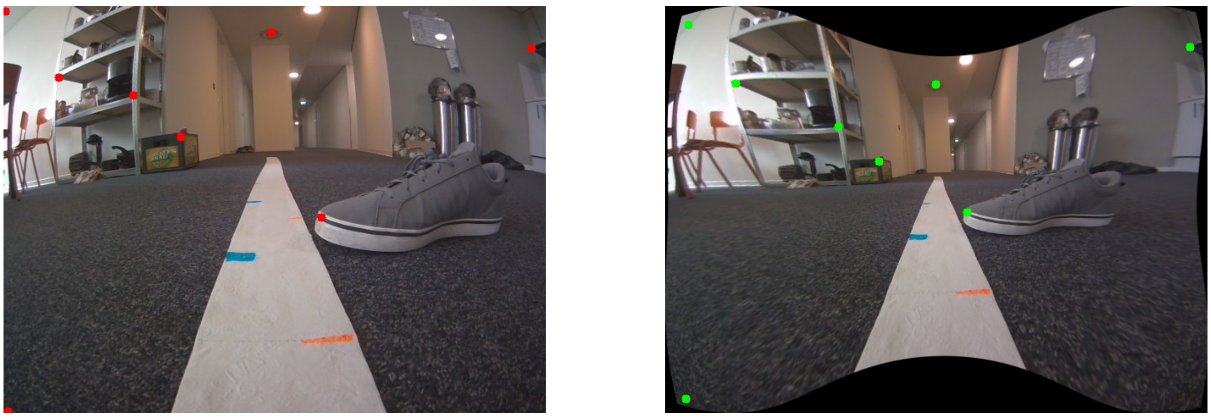
The Time of Flight sensor is one of the tools used to measure the distance between the sensor and any obstacle that is in its field of view. The principle of the ToF sensor is based on the time it takes for light beams to travel towards the object, reflect and return back to the sensor, hence the *time-of-flight*. In our implementation we use it to detect the distance between the shoe and the Duckiebot. The measurement made from the initial estimation of shoe distance through computer vision techniques mentioned in the section below, it is further reinforced with the distance measured from the ToF sensor. It provides a relatively more reliable source of information of distance. A drawback of the ToF sensor is that its range is limited to 1.2m, and objects further away than that are not detected. A way we overcame this is by estimating the distances to objects visually through the images, and then when the Duckiebot is close enough it switches to the measurement done by the ToF sensor.

#### 3.4.2 Visual distance estimation

In a pinhole camera, projections can be easily computed and that way we could estimate the relative position of an object just by the visual input. However, our real-world camera introduces distortions that make this process more challenging. First, we need to undistort the image, then we can measure the shoe height in pixels and the angle from which we are observing it. Finally, we will be able to compute the distance.

We can use the widely available OpenCV toolkit to transform and analyze the image. The first step to solve this issue is to undistort the image so as to have a more homogeneous focal point across the whole image. As displayed in Fig. 2, the undistorted image preserves the lines straight. On this image, using the bounding box of the shoe, we can get the depth projection by the following expression:

$$\text{projection}_x = K \frac{\text{shoe\_height}_m m}{\text{shoe\_height}_p x}$$



(a) Raw image. (b) Undistorted image.

Figure 2: Side by side comparison before and after undistortion.

Where $K$ is a parameter that encompasses several constants, such as the focal length of the camera. We obtained this parameter by gathering experimental data in the range that the object detector is capable of spotting the shoe and then adjusting from a value that works better in the range that we expect the robot to operate. This is illustrated in Fig. 3.

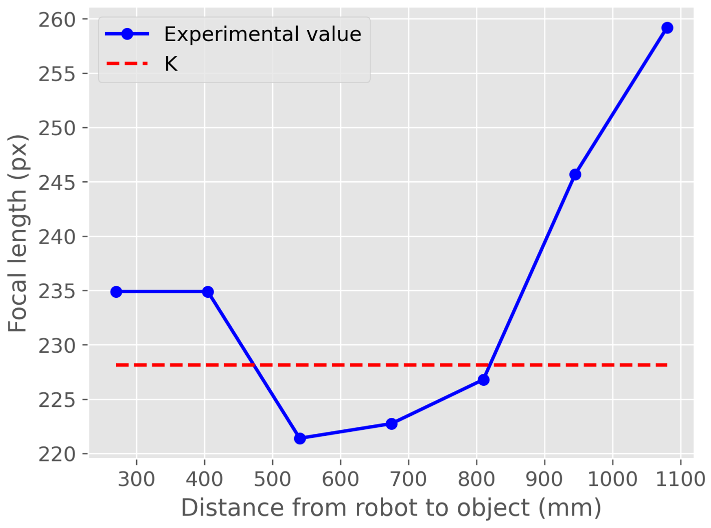


Figure 3: Projection constant: Experimentally obtained values, in blue. Selected value, in red.

The reason we are able to proceed this way is that we can be certain of the height of the shoe, which doesn't change due to the robot being always on the floor. This is not true for the width of it, since it varies depending on the rotation of the shoe. Thus, we cannot follow the same procedure to get the projection on the second axis. However, since the undistorted image preserves the angles. We can seize this property to measure the angle from a vertical line that splits the frame in two and a line from the center bottom-most point to the centroid of the shoe. This angle, $\varphi$, is the same as the projected on the floor plane. Finally, we can calculate the final estimation applying the following identity:

$$\text{distance} = \frac{\text{projection}_x}{\cos\varphi}$$

## 3.5 Odometry

Our robot needs to have an idea on its position and orientation over time to properly fulfill its tasks. We have two sources of odometric information that we merge together using a Kalman filter. This technique models the system behavior over time, combining several sensor information and taking into account the reliability of the measurements. The process involves two main steps: prediction and update, as expressed in the following formulas:

$$\begin{aligned}
&\text{Prediction Step:}\\
&\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_{k-1}\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1}\\
&\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1}\\
&\text{Update Step:}\\
&\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\left(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k\right)^{-1}\\
&\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\left(\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1}\right)\\
&\mathbf{P}_{k|k} = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{P}_{k|k-1}
\end{aligned}$$

For the first, it takes the current state, $\hat{x}$, of the system and predicts the next state with the input measured by the sensors, $u$. The way these vectors affect the next state is embodied within

$F$ and $B$. The update state happens when new measurements allow to refine the predicted state. Throughout the process, a covariance matrix,$P$, keeps track of the uncertainty in the estimated state, regarding the noise of the readings, $Q$ and $R$. In our case, we have frequent readings provided by the encoders in the wheels and sometimes position estimates when an April tag is detected.

### 3.5.1 Encoder odometry

Encoders are the most basic sensors to estimate the pose of our robot. By analyzing the data they provide, we can estimate the distance travelled and the orientation change. With an almost constant frequency, they publish the total ticks, so we can get the increment from reading to reading and calculate the distance travelled in each axis with the following expressions:

$$\begin{aligned}
\Delta\phi_k &= N_{rot} \cdot \Delta t_k \\
d_{l/r,k} &= R \cdot \Delta\phi_{r,k} \\
\Delta\theta &= \frac{d_r - d_l}{2L} \\
d_A &= \frac{d_r + d_l}{2}
\end{aligned}$$

Where $N_{rot}$ stands for the ticks, $t$, for a whole wheel rotation; $R$ is the wheel radius; $d$ is distance travelled by a wheel; $L$ is the baseline of the wheels. Finally, we can calculate the increment in each axis by:

$$\begin{aligned}
\Delta x =& d_{A,k}\cos(\theta_k) \\
\Delta y =& d_{A,k}\sin(\theta_k)
\end{aligned}$$

### 3.5.2 April Tags

Due to the variability and inaccuracy that some measurements provide, we need another source of precise information relating to the location and orientation of the duckiebot. The April Tags library ensures a good prediction of the distance a tag from our agent. In detail, by placing the Tags in the four cardinal directions of our moving environment, the April Tag node extracts the camera and distortion coefficients and creates a Detector structure for identifying the tags. By subscribing to the camera node, it processes the image through the Detector and outputs the (x,y,z) coordinates of the tag in terms of the camera plane as well the the tag's orientation. Because the y coordinate describes the distance of the tag from the floor's plane, and we are only interested in the distance of the robot to the tag in the (xz) plane, we calculate their Euclidean distance as:

$$d_{AT} = \sqrt{x^2 + z^2}$$

Since we want to estimate the robot position in the context of the reference map and not itself, we need to do some trigonometry. From the camera, we can measure the angle at which the tag is being observed,$\varphi$ , and the distance to it,$d$ , as mentioned in section 3.4.2. We can also obtain the angle from the perspective of the tag,$\psi$ , out of the orientation matrix of the April Tag. With these three values we can estimate the robot's using the following expressions:

$$\theta = \varphi + \psi + C \tag{1}$$

$$\Delta x = d \cdot \sin\varphi \tag{2}$$

$$\Delta y = d \cdot \cos\varphi \tag{3}$$

The detected tag contains an ID, from which we can know the absolute position of the tag, as well as its orientation, and select the constant, $C$, as well as know the robot's position. For example, for a tag placed in (1,1) with orientation -180º. If we were to observe it from an angle of 30º at a distance of 2:

$$\begin{aligned}\theta =& 30 + 150 - 90\\ \Delta x =& 2 \cdot \sin 30\\ \Delta y =& 2 \cdot \sin 30\end{aligned}$$

The position of the robot would therefore be: (0, -0.73, 90º)

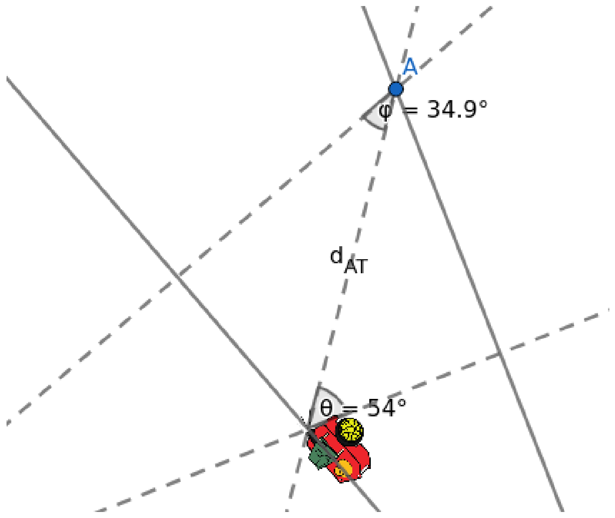


Figure 4: Position of the robot relative to the April Tag.

## 3.6 Motion control

The movement of the robot has been designed to follow a sequence of goal positions in terms of global coordinates. Motion control is crucial to ensure that the Duckiebot stays on course and properly steers towards the goal position.

To address this challenge, we have opted to implement a PID controller. However, in contrast to a normal PID controller, there are two errors that the node responsible of motion control has to correct. One is the distance between the goal position and the current position of the robot. The other is the difference between the desired angle of the goal position and the current angle. Moreover, there are two different outputs of which the PID can actuate on to reduce the error: linear and angular velocity of the motors.

The first approach taken was to basically have two PID controllers running in parallel. While one controls the angular velocity to fix the desired final orientation the other one adjusts the velocity to reduce the distance between points. However, this implementation only works when the desired angle happens to be on the direction to the goal position. Otherwise, both PID increase the error of the other controller causing both errors to diverge.

To fix said problem, the final design implements a state machine to change the behaviour of the PID depending on the situation of the robot in respect to the goal position. The following states are the following:

- **Adjusting orientation.** If the robot is far away from the goal destination and it is not aligned in the direction of the goal position the PID is set in this state. When this is the case, the error to minimize is the angle between the goal position and the current position and it actuates on the angular velocity.

- **Closing the gap.** Once the robot is orientated to the goal destination then it only needs to go forward to get closer to the point. Therefore, the actuating value can be linear velocity and the input error used can be the distance.
- **Fixing theta.** The state changes once again when it is close to the destination point. When the robot is close enough to the desired point all it has left to do is to adjust the desired theta for the final position.
- **Idle.** When all the previous conditions are met, the robot is considered to be close enough to desired positions and the linear and angular velocity for the motors are set to 0.

All the thresholds to change from one state from one state to another are given an Hysteresis range. Meaning that to change from one state to the next one a small threshold is given. However, to get back to the previous state the threshold value is higher. This hysteresis avoids oscillations of the PID due to constant switches from one state to the other. A visualization of said states is shown in the following Figure 5.

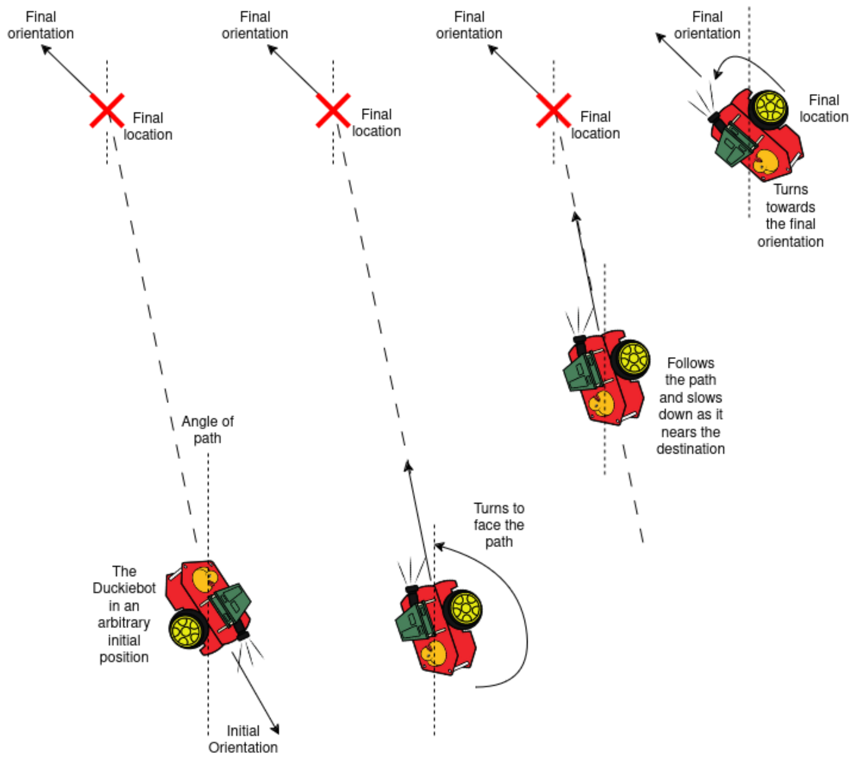


Figure 5: The PID Control executing its plan of action to reach the destination

As it has been mentioned, the PID controller relies on the current position and orientation of the robot in relation to the global coordinates. This information is obtained from the odometry node explained on the previous section.

Going into more details of the controller, the PID relies on the adjusting of the three constants $k_p$, $k_i$ and $k_d$. Each constant stands for proportional, integral and derivative as they scale said correction of the error as seen in the following equation.

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)d\tau + k_d \frac{de(t)}{dt}$$

Nevertheless, in our implementation the error signal is not longer continues but rather it has been discredited and quantified. Moreover, a new value is given every time an update in odometry is received, which might not be a constant rate. Therefore, the resulting PID follows the next equation.

$$u_k = k_p e_k + e_{int,k-1} + e_k \Delta t + \frac{(e_k - e_{k-1})}{\Delta t}$$

Now the $d\tau$ is replaced by the difference in time between the received odometry message and the previous one. Also for the integral part calculation a new variable is introduced which accumulates all previous errors. Lastly, for the derivative the previous error is saved so that it can be subtracted with the current one and obtain the rate of change.

## 3.7 Path planning

From the previously mentioned nodes, the position of the robot and the shoes can be obtained. With said information and after selecting a new node called orchestrator sends the trick to perform and the shoes of interest to another node called path planner. The path planner node, using the knowledge of the coordinates of the shoes, generates a set of poses that the Duckiebot will follow using trigonometric functions. These poses are first published to RVIZ to visualise the path that it will take and then relayed one by one to the PID controller iteratively until all the poses of the path have been executed. When the path planner first receives the decision it finds the point that is closest the list of points in the path and chooses that as the starting pose.

### 3.7.1 Eight figure path

Considering the two points of the shoes to be $(x1, y1)$ and $(x2, y2)$, we calculate the midpoints of these points and also the distance to the midpoint of the shoes. The formula used to calculate the path for the figure eight is given by the following formula which describes the path shown in Figure 6

$$x = (distance/2 + radius) * cos(\theta)$$
$$y = radius * sin(2\theta)$$
$$x_{path} = midx + x * cos(\theta) - y * sin(\theta)$$
$$y_{path} = midy + x * sin(\theta) + ysin(\theta)$$

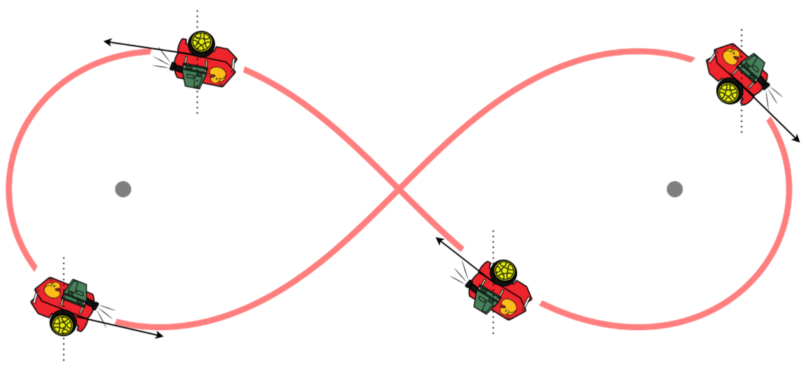


Figure 6: The Duckiebot on an eight figure path

### 3.7.2 Circular path

Using the same convention as mentioned before, the equations for the circle shown in Figure 7 are given by the next equation

$$x_{path} = midx + (distance + 0.5) * cos(\theta)$$
$$y_{path} = midy + (distance + 0.5) * sin(\theta)$$

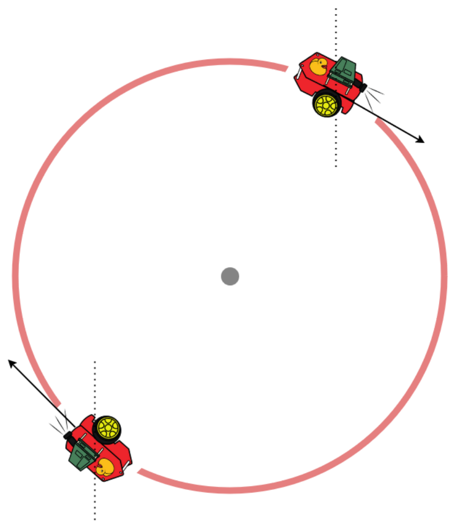


Figure 7: The Duckiebot on a circular path

The other two tricks include following a persons shoes and rotating in place in accordance to the position of the shoe.

## 3.8 ROS graph and specific functionality of the nodes and topics

In this section figures are provided showing the architecture of the final implementation. Figure 8 is the output when executing RosGraph while Figure 9 shows a simplified diagram of the most important node and elements.

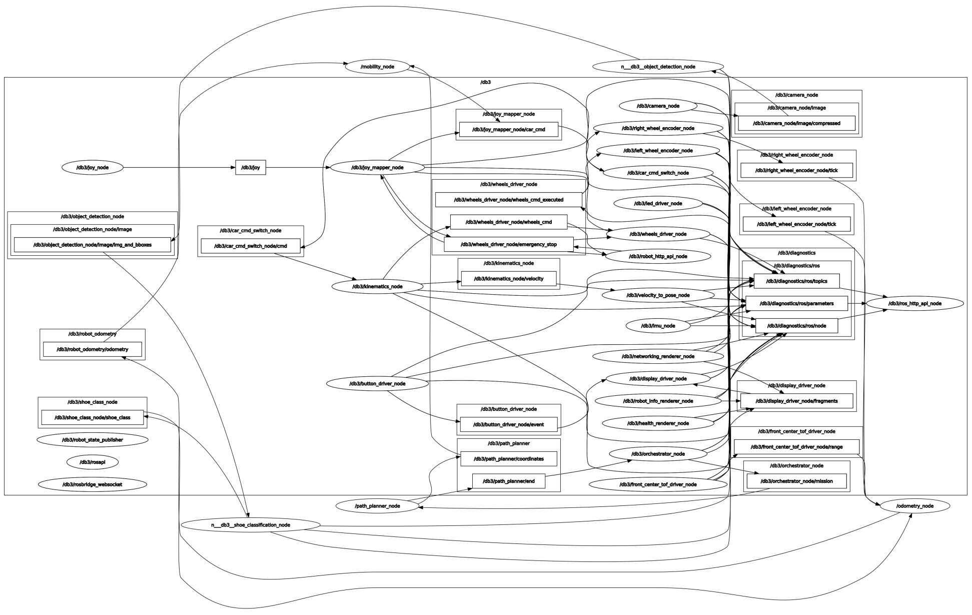


Figure 8: RosGraph output of the final implementation

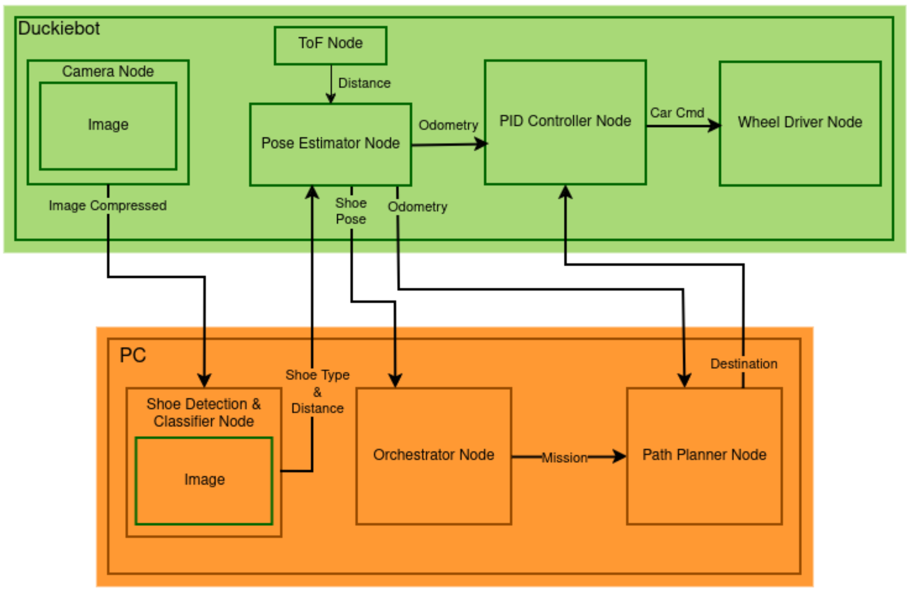


Figure 9: Distribution and Architecture of the relevant Nodes of the application

## 4 Deployment details

All the nodes were created from scratch using the basic duckietown template. The CNN for object detection and classification was decided to run locally on a laptop a single node instead of two to minimise latency. Running it on the laptop also reduces the computational intensity on the Duckiebot and improves throughput of the topic processing the frames. The distribution of the various other nodes can be found in image 9. The nodes like odometry and PID were run remotely on the Duckiebot to reduce the latency of communication since they involved real time tasks which need fast feedback. The location of the ROS core also plays an important role in the performance of the application since all the nodes use this as a point of communication with other nodes. The ROS core in our application runs remotely on the Duckiebot. The ROS core is initialized by the launch scripts that run roslaunch while launching their respective nodes

## 4.1 Performance metrics

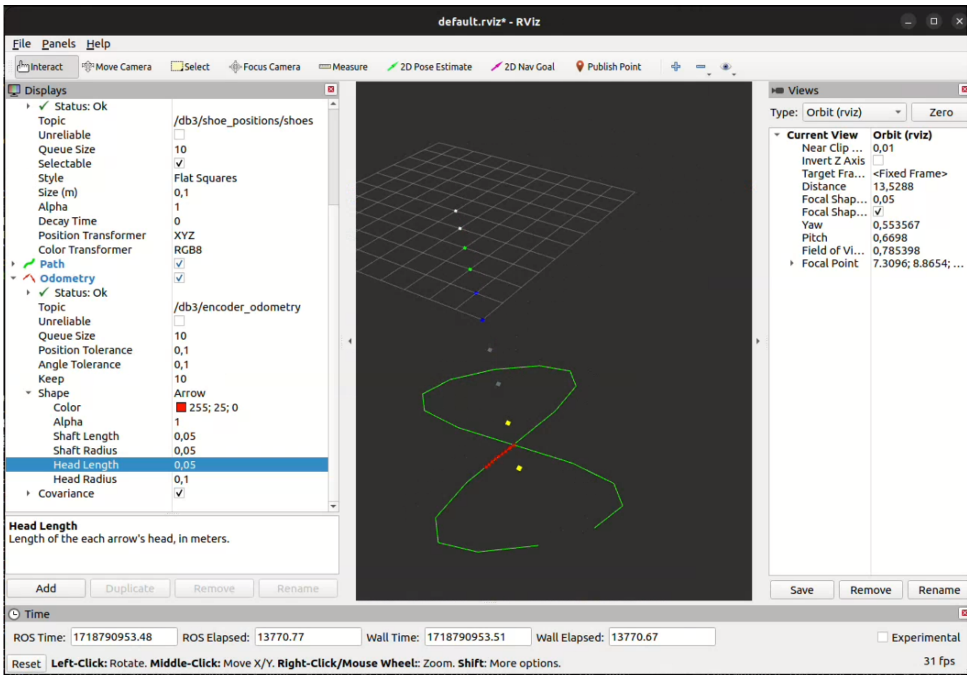
| **Parameter** | **Details** |
|---|---|
| Timing | Real-time processing at 30 FPS |
| Throughput | Handle up to 5 frames per second |
| Confidence of classification CNN | 1.0 |
| Distance Estimation of shoes using camera | Measures distances up to 1 meters with 80% accuracy |
| Distance Estimation of April tags using camera | Measures distances up to 1.5 meters with 95% accuracy |
| Obstacle distance using ToF | Detects obstacles up to 1.2 meters |

The high measurement distance of April tags and its accuracy is attributed with the fact that the dimensions of the April tags are always a constant 11x11 cm and because of the already implemented *lib-dt-apriltags* library. On the contrary the shoe distance estimation algorithm is a rough estimation implemented on our own and also due to the fact that all the shoes are of different dimensions. The fact that the bounding box sizes change every time introduce certain measurement errors.

## 4.2 Testing and simulation

In order to enable testing even when not all the nodes were finished or published RVIZ was used. RVIZ allows visualization of certain types of topics. Moreover, this topics can be mocked which allows simulation of certain behaviours without relying on the actual implementation of other nodes.

The path planning node sends a Path message containing the poses the robot will take for a particular manoeuvre to RVIZ to visualise the plan the Duckiebot will execute. This was done successfully with the Duckiebot mimicking its position in the virtual environment where it is able to plot a point cloud of the shoes and receive and generate the path from the planner node too.



```
Status: Ok
Topic                   /db3/shoe_positions/shoes
Unreliable              ☐
Queue Size              10
Selectable              ✓
Style                   Flat Squares
Size (m)                0,1
Alpha                   1
Decay Time              0
Position Transformer    XYZ
Color Transformer       RGB8
Path                    ✓
Odometry                ✓
  Status: Ok
  Topic                 /db3/encoder_odometry
  Unreliable            ☐
  Queue Size            10
  Position Tolerance    0,1
  Angle Tolerance       0,1
  Keep                  10
  Shape                 Arrow
    Color               🟥 255; 25; 0
    Alpha               1
    Shaft Length        0,05
    Shaft Radius        0,05
    Head Length         0,05
    Head Radius         0,1
  Covariance            ✓

Head Length
Length of the each arrow's head, in meters.
```

```
Type: Orbit (rviz)
Current View       Orbit (rviz)
  Near Clip ...    0,01
  Invert Z Axis    ☐
  Target Fra...    <Fixed Frame>
  Distance         13,5288
  Focal Shap...    0,05
  Focal Shap...    ✓
  Yaw              0,553567
  Pitch            0,6698
  Field of Vi...   0,785398
  Focal Point      7.3096; 8.8654; ...
```

```
ROS Time: 1718790953.48   ROS Elapsed: 13770.77   Wall Time: 1718790953.51   Wall Elapsed: 13770.67   ☐ Experimental
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.   31 fps
```

Figure 10: Simulation of the eight path on RViZ. The yellow points on the graph represent the positions of the two shoes and the red arrow represents the path that the Duckiebot is taking.

However this is very far from implementation in a real world environment. There are many conditions like lighting, surface and obstacles that may hinder performance and the working of the robot. In our testing of the PID controller different values had to be tweaked for the surface of a table and the surface of a carpet as the latter induced slipping of the wheels and resistance to motion. The simulator provides controlled environment and any testing needs to be done with external environment taken into consideration.

## 5 Validation

| ID | Satisfied | Remark with specific details |
|---|---|---|
| R1 | Yes | The April tag ID's could be identified correctly with high accuracy in reading and far distances |
| R2 | Yes | Shoes are identified within a range of 1.2m |
| R3 | Partially | Shoes with a strong color palette are identified correctly while others are often misclassified |
| R4 | Yes | The robot's position and orientation is estimated using a Kalman Filter |
| R5 | No | The robot can detect obstacles but avoidance was not implemented |
| R6 | Yes | Distance to shoe could be calculated using a mix of computer vision and trigonometry |
| R7 | Yes | Using April Tags we acquire a better estimation of the robot and using computer vision calculate the position of the shoe done in R6 |
| R8 | No | Since R9 was not implemented, this was also not carried out. |

**Table 3 continued from previous page**

| | | |
|---|---|---|
| R9 | No | Although we implement the path around shoes, we do not take this requirement into consideration |
| R10 | Yes | A mix of PID and Kalman filters enable the Duckiebot so that it does not deviate from its path |

Table 3: The list of the requirements satisfied

In table 3 we have vaildated all the requirements from table 1. Requirements R8 and R9 couldn't be validated since we assume that the person will be standing with his legs wide apart, enough for the duckiebot to pass through. Requirement R5 was also not implemented and validated as finding obstacles and dynamically re-configuring the planned paths was found to be a little complex.

## 6 Conclusions

The final deployed function enables the Duckiebot to detect, classify, and follow specific shoes, while performing complex maneuvers based on the distance between tracked shoes.

- This project can be extended by integrating more advanced functionalities such as voice commands for interaction, more detailed localization techniques, and enhanced obstacle avoidance mechanisms.
- Develop algorithms that allow the Duckiebot to dynamically adjust its trajectory upon detecting an obstacle.
- Implement additional sensors such as LIDAR, ultrasonic, or infrared sensors to enhance the existing vision-based system. This would provide increased spatial awareness to the Duckiebot.
- It has been seen that for different surfaces the PID coefficients will have to be tweaked. If the values are chosen by trail an error on one surface then they might not be useful on a more slippery or rough surface. However, if the odometry is reliable this manual process could be automatized so that it detects the best PID coefficients while the robot is running.

Details of the code repository and any specific comments to use the codebase can be found in the `Readme.md` file in the repository[1].

[1]https://github.com/Serge916/Shoe-san.git