# Project Overview (Rosh)

# This project aims to control the turtlebot to follow a straight line by using the information observed from its RGB-D camera and moving it perpendicular to the square object. This project utilises ROS in Ubuntu 18.04, testing the turtlebot in a Gazebo environment, and the code was tested and written in MATLAB 2021b. The square was made with a custom face of a grid layout on each corner and an image of a car in the middle. We have had multiple iterations of the square, as you can see in the next slides. We then implemented and used image processing, testing with SURF and KAZE feature detection to utilise accuracy and maintain computing cost. A PD controller was also implemented to correct the turtlebot towards the goal, accounting for any sudden changes in acceleration.

# Project Timeline (Rosh)

This was the project timeline implement to complete the project in time. The development of image processing begun at week 6, concurrently with the square model used for the project.

# Methodology (Rosh)

The methodology of the project begins with three sections, ROS initialisation, Image processing and PD Controller. Firstly, the ROS environment was initialised, subscribing to topics such as rgb, depth and odometry for the Turtlebot localisation and RGB-D camera. A ROS node, called driving, was also created to publish control commands for the Turtlebot. Another part was to also build a gazebo environment for the Turtlebot, design the square and its unique pattern, and edit a launch file for these models to be initiated in Gazebo.

The image data extraction and processing begin with the turtlebot autonomously rotating anti-clockwise in its position, until it finds the square pattern. It will eventually detect the pattern using SURF feature detection, using an image of the pattern given, and RGB and depth data coming from the RGB-D sensor, then they will both be matched and paired. After it detects four feature pairs from the image and data, it will estimate geometric transformation between the matched pairs and remove outliers and obtain inlier feature pairs, to improve accuracy of feature alignment. Finally, a point cloud is formed from the feature points, a plane is fitted in the point cloud and intersection points are found between the feature plane and the turtlebot plane. This will then provide information for the turtlebot’s orientation and the coordinates of intersection points, where the angle to the closest intersection point is calculated from the robot’s position, calculating the pose relative to the square.

Lastly, a PD controller was implemented to calculate errors between the orientation of the Turtlebot to the target and create adjustments for its movement and control. We set two parameters called Kp = 0.5 for proportional gain, and Kd = 0.25 for derivative gain. The control input was derived from these parameters to adjust the robot’s angular velocity, allowing it to align with the square and correcting any sudden changes with acceleration.

# ROS Graph (Matthew)

This ROS graph provides the graphical flow and structure of how ROS is utilised in this project. As seen in this image, the driving node made for this project publishes data for cmd\_vel, which is the turtlebot’s control. It then communicates to Gazebo, where it subscribes to three topics: odometry, RGB-D camera’s raw image depth and the image. Finally, ROS communicates with MATLAB’s ROS network.

# Experiments (Matthew)

# Project Issues (Daniel Lo)

# Project Demo Video and Results (Daniel Lo)

# Final Results (Rosh)