# Project Overview (Rosh)

# This project aims to control the turtlebot to follow a straight line, perpendicular to the square model, by using its on-board RGB-D camera. This project utilises ROS in Ubuntu 18.04, testing the turtlebot in a Gazebo environment, and the code being developed in MATLAB 2021b. The square was made with a custom face of a corner grid layout on each corner and an image of a car in the middle. We have had other iterations of the square model, as you will 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.

# Project Timeline (Rosh)

This was the project timeline we implemented to complete the project in time. We initially begun with defining the project environment and learning the turtlebot controls and simulation. Then we began development for image processing at week 6, concurrent with the square model used for the project. We then begun with localisation and ROS development within MATLAB such as publishing nodes and subscribing to topics. Finally, we finalised the code at Week 9-10, with the demo video finalised.

# Methodology (Rosh)

The methodology of the project begins with three sections, ROS initialisation, Image processing and PD Controller. First, 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.

The image data extraction and processing begin with the turtlebot autonomously rotating in its position, until it finds the square pattern. It will then 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)

For experimenting the different methods of feature detection, we looked into SURF and KAZE feature detection algorithms. Using KAZE provided more accuracy and reliability, we opted for SURF detection due to its computational efficiency and speed. This was due to the issues we found in the developing the project, which we will talk about in the next slide.

The MATLAB profiler was utilised to provide information such as memory usage, total run time of the project and function usage. The profile provided information on how we can analyse and optimise the performance of the code. It was able to show the memory usage, which was vital for us when running the simulations in a virtual machine due to limited memory.

For the square’s pattern, we were able to test and develop different pattern and features for the RGB-D camera and feature detection testing. Initially we developed our first pattern using a full-face grid layout and having each of the four corners have a unique colour of red, green, blue, and black, and having inlier squares of the same colour pattern, as seen in the image on the right. However, afterwards we opted for a more unique pattern of still having a grid layout, however only for the four corners where each grid corner has a unique pattern with the same colour scheme, and an image of a car in the middle. We have also experiment with a custom environment of a simple room with walls and a floor layout for the project.

# Project Issues (Daniel Lo)

These were some of the issues we faced in the project. The consistent issue we faced was that running these simulations and MATLAB in a virtual machine showed us that some of us lacked the computational power to run them consistently, making the development inconsistent across all team members. This was one of the reasons why we chose SURF feature detection, as it required less memory to run and test, compared to KAZE.

Another issue found was the orientation calculation issues due to the complexities of sensor fusion. The orientation was derived from the odometry and the camera’s RGB images. Integrating the data required mathematical transformations and geometric computations, which added complexities to the project.

# Project Demo Video and Results (Daniel Lo)

Here we have the project demo video playing, to show the turtlebot following a straight line by using the information observed from its RGB-D camera, perpendicular to the square object. We also have here the profile summary generated from MATLAB Profiler during the demo of the project. This shows our total run time of around 71 seconds, the memory usage and displaying the function usage in the duration of the demo.

\*Pause 5 seconds

Thanks for Watching