CODING ROBOT BEHAVIOURS BASED ON ROS SENSOR INPUT

A custom wheeled robot in Gazebo's postoffice world (Project undergoing)

Skills learned:

Beginner

- Using the camera/lidar on the robot as well as local webcam and wrapping it up in ROS
- Inserting custom computer vision code (OpenCV) and wrapping it up in ROS
- Using ROS nodes in different configurations

Intermediate

- Creation of a custom robot model in Gazebo (URDF and SDF)
- Creating a state machine for multiple robot states in SMACH

Advanced

- Modularising robot behaviour by wrapping the state machine in a ROS Action Server
- Face recognition and wrapping it up in ROS

Essence of the project

In this project, a wheeled robot locates a gate and then proceeds to drive through it. This serves as an introduction to **coding robot behaviours** based on **what it observes**.

[Intro driving gif]

This project showcases a custom wheeled robot in Gazebo's postoffice world.

- Creation of a custom robot model in Gazebo (URDF and SDF)

There are many cases where a robot is required to react to its environment, and they do this via sensors (camera, lidar) to 'gather' information, and then 'decide' what to do with the information, before proceeding with an 'action'.

We make use of a state machine to be our 'decision-maker' between the sensor input and specific motions of the bot. SMACH is a simple Python implementation of state machines, and for the purposes of developing various usecases, you can get up and running easily.

- Creating a state machine for multiple robot states in SMACH

ROS is used as a **communication protocol** to gather the desired sensor input, and to execute the robot behaviours, and since ROS wraps our code, it helps us make the code **portable** to a real robot. This project was in fact ported to a Turtlebot (real robot), where ROS nodes were simply detached from the simulator and attached to the Turtlebot3 hardware.

- Using the camera/lidar on the robot as well as local webcam and wrapping it up in ROS
- Inserting custom computer vision code (OpenCV) and wrapping it up in ROS
- Using ROS nodes in different configurations

Finally, a central decision planner is used in preparation for scaling up the network. To do this, a ROS Action Server wraps the state machine.

- More on Action Servers with ROS and SMACH

With further development, I managed to integrate a pattern recognition algorithm (using haar cascades ML and wrapping it in ROS). It recognises faces via webcam and robot camera. Further development could make the robot recognise different gestures and react accordingly.

- Haar cascades and wrapping it up in ROS

Questions

<u>Why use Gazebo if I can test directly on the robot?</u>

The use of a state machine and ROS helps us develop a **network of nodes** for the different components of the system. You could use both a real and a virtual robot to test out this code, but this project **relies less on performance** and more on **wrapping the functionality with ROS**. When I used the Turtlebot (a real wheeled robot), I could execute tests directly on the bot, however, our main bottleneck with ROS nodes had to be resolved before calibrating any type of motion. The lack of a testing space slowed down our project to a halt, especially since multiple developers need a common framework for tests, development, performance optimisation, etc.

<u>I can wrap inputs and outputs with ROS, but why create a state machine?</u>

A state machine will help build up a network of usecases that are:

- sometimes unforeseeable: changes in lighting might affect the robot so much that we choose to code two different behaviours based on the lighting conditions.
- not only sequential but simultaneous, and sometimes triggered randomly, or listed in order of priority: if there's an obstacle on the robot's path to the gate, it needs to detect and drive around it before focalising on the gate.

[TurtleSim Gif: Part of a tutorial to learn to make a state machine with TurtleSim]

A state machine allows us to coordinate the transition between different usecases. You could as well use a C++/python script to analyse camera information and to code up a sequence of actions. Initially, I ***did this too***. In fact, you can code on a microcontroller in this way, with good understanding of exceptions, interrupts, global and local scoped variables. It might depend on what you expect your **speed of prototyping** to be.

- Creating a state machine for multiple robot states in SMACH

The final reason for state machines is to centralize the decision-making processes of our robot infrastructure. Take a robot that has 6 cameras giving it a 360° view. Imagine now that you want to select which cameras should be active and deactivated, based on system resources. If each camera is a node of the network, using a central decision-maker becomes incredibly useful.

- Face recognition and wrapping it up in ROS
- More on Action Servers with ROS and SMACH

<u>Why is this network centralized?</u>

Robot autonomy can be achieved with various levels of implementation.

- **Functionality**: The functionality of the robot can be tested with various virtual simulators. It is the case here that OpenCV code is tested in a Gazebo simulator, using a custom robot model and camera.

Infrastructure: Transferring this functionality to a real robot. This use of ROS If our inputs and outputs need to network over a communication protocol. Most often, our robots must transmit information over wifi to a basestation. Such information can be used to issue commands to the robot remotely and assist the robot, but also can for observational purposes, to record data issued by the robot. These can be done in different configurations:

- Server-client: activated by client
- Request-response:
-