V1.7 | Last updated: 25 october 2020

BOT-DRIVES-TO-WAYPOINT PROJECT

Objective: Setup a custom car in Gazebo, that drives autonomously through a gate.
Duration: 1.5 weeks. Depends mainly on time spent in prep and setup time.
Learning outcomes: Merging technologies with ROS and the foundations of behaviour planning.

STEP 0	**| PREP**	**| 2 days**
STEP 1	**| SETUP**	**| 2 days**
STEP 2	**| PROTOTYPE**	**| 2 days**
STEP 3	**| DEMO**	**| 1 day**

INTRODUCTION

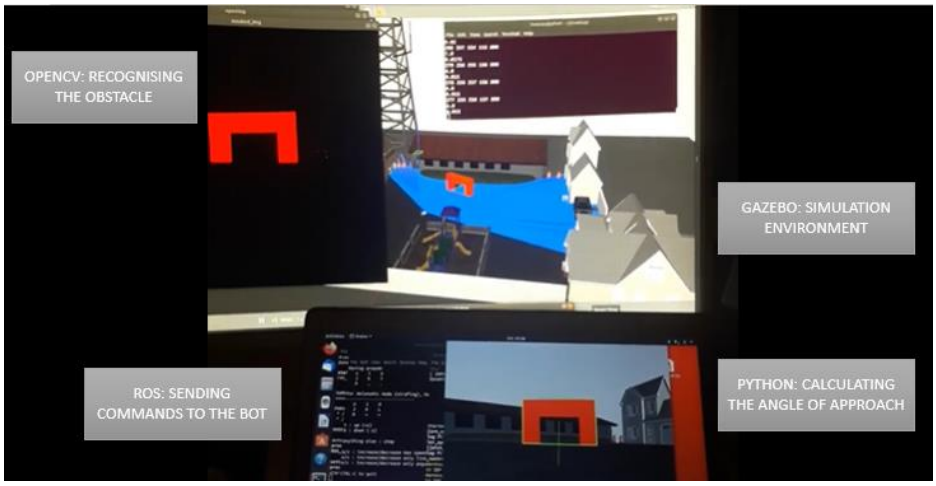
This project uses a wheeled bot in Gazebo and programs it to detect and approach a gate. This project, for me, was a primer in the use of a simulator, **Gazebo** with a programmable middleware, **ROS** in a field that interests me, **waypoint following for robotics**. This project is therefore clearly linked to autonomous drones, as they also can navigate by following waypoints. However, the specific usecase here is about:

- **coding bot behaviours within the Gazebo simulator based on what the bot observes**,
- while also creating a modular environment intuitive enough for other functionality to be added later.

Various functionalities are designed and we will be able to explore them in greater depth in a series of tutorials. This page, instead, looks at what was achieved in the project. See these pages on the site for introductory information.

- *Robot behaviour planning*
- *Basic computer vision for robots*
- *Linux and ROS background*
- *Creating applications with ROS*

The final demo is accessible at https://www.youtube.com/watch?v=rdJYIxeUt-o . It incorporates the technologies shown below.



ROS Textbook - https://drive.google.com/file/d/147l-EPXTe4QW_H9m5FHB2yH3JsMd5oMQ/view?usp=sharing (request access first)

Note: this document was last updated in October 2020. Please signal any issues at thomas.carstens@edu.devinci.fr .

1. Why ROS?

Usefulness of:

- A networked environment
- A Middleware


1. Niveau Débutant

1. Installation et Configuration de Votre Environment ROS
Ce tutoriel vous guide à travers l'installation de ROS et la configuration de votre environnement ROS pour votre ordinateur.
2. Navigation dans le Système de fichier ROS
Ce tutoriel introduit les concepts du système de fichiers ROS, et couvre l'usage des commandes de roscd, rosls,et rospack
3. Créer un package ROS
Ce tutoriel présente l'utilisation de roscreate-pkg ou catkin pour créer un nouveau package et rospack pour lister les dépendances des packages
4. Construire un package ROS
Ce tutoriel détaille la création d'un package ROS à l'aide d'un ensemble d'outils ROS.
5. Comprendre les "nodes" ROS
Ce tutoriel introduit les concepts de ROS graph et l'utilisation des outils en ligne de commande roscore, rosnode, et rosrun
6. Comprendre les Topics ROS
Ce tutoriel introduit les concepts de Topics sous ROS ainsi que l'utilisation des outils en ligne de commande rostopic et rqt_plot.
7. No Title
No Description

Project

Objective: Have the car drive through the gate, autonomously.
Duration: 1 week. (depends on how fast you do the ROS tutorials)

Why use Gazebo if I can test directly on the robot?

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.

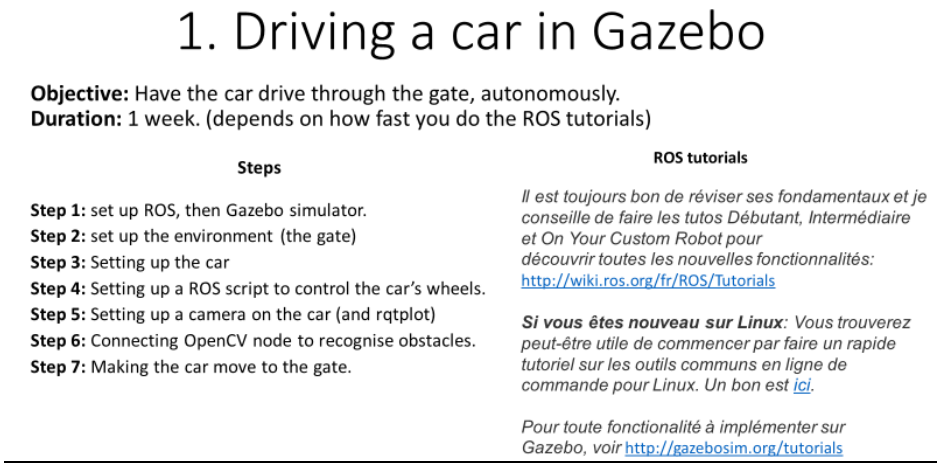
ROS is used as a **communication protocol** to gather the desired sensor input, and to execute the robot behaviours, and since ROS wraps our code, it helps us make the code **portable** to a real robot. This project was in fact ported to a Turtlebot (real robot), where ROS nodes were simply detached from the simulator and attached to the Turtlebot3 hardware (this project can be seen here https://www.youtube.com/playlist?list=PL1ezH_EwB1WV0NmtxSxzI_leFI_YRmhL7)

- 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

PRELIMINARY

Installation of Ubuntu (dual boot, preferably), of ROS, and finally of Gazebo.
The Ubuntu and ROS steps are outlined here:
https://www.youtube.com/playlist?list=PL1ezH_EwB1WV0NmtxSxzI_leFI_YRmhL7

It is always better to be prepared when approaching a new program. I advise you go through the ROS tutorials, UNIX if necessary, and Gazebo.



1. Driving a car in Gazebo

Objective: Have the car drive through the gate, autonomously.
Duration: 1 week. (depends on how fast you do the ROS tutorials)

Steps

Step 1: set up ROS, then Gazebo simulator.
Step 2: set up the environment (the gate)
Step 3: Setting up the car
Step 4: Setting up a ROS script to control the car's wheels.
Step 5: Setting up a camera on the car (and rqtplot)
Step 6: Connecting OpenCV node to recognise obstacles.
Step 7: Making the car move to the gate.

ROS tutorials

Il est toujours bon de réviser ses fondamentaux et je conseille de faire les tutos Débutant, Intermédiaire et On Your Custom Robot pour découvrir toutes les nouvelles fonctionnalités: http://wiki.ros.org/fr/ROS/Tutorials

***Si vous êtes nouveau sur Linux**: Vous trouverez peut-être utile de commencer par faire un rapide tutoriel sur les outils communs en ligne de commande pour Linux. Un bon est ici.*

Pour toute fonctionalité à implémenter sur Gazebo, voir http://gazebosim.org/tutorials

Now, we can get started!

Contents

DESIGNING A WHEELED BOT: BACKGROUND

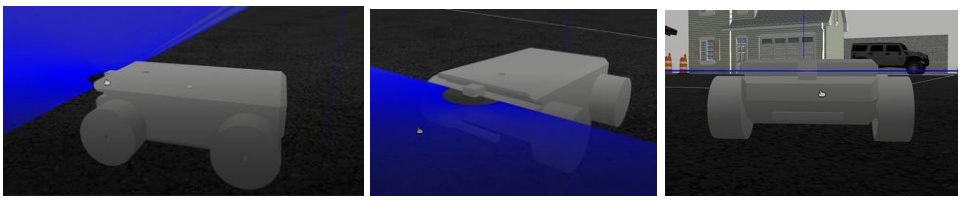
First we test the URDF in an empty simulator.

The Unified Robotic Description Format (URDF) is an XML file format used in ROS to describe all elements of a robot. I followed a tutorial provided by the ROS documentation to create the URDF format, and another from the Gazebo documentation be able to convert it to SDF (XML file format that is readable by Gazebo).

Read the theory to better understand what URDFs consist of. But don't do it yourself 😊 because there are too many mistakes in the textbook!

- **Modelling the robot base:**
- **PG106 OF ROS TEXTBOOK**

Error 1: Main difficulty: the tutorial omits the first few <include> to link different xacro files in between them. As a result: parse errors ; the model will not be able to spawn. See my github model for those lines… or just figure out how to copy my model's code.



As you can see, the front two blocks house the lidar sensor (from which emanates the blue rays) as well as the RGB camera (elongated block).

Note: make sure you understand the Gazebo tutorials linked above, as well as the ROS textbook explanation.

DESIGNING A WHEELED BOT: PRACTICE

I used the ROS manual for this, which you can find in our ROS Resources Page.

You can easily get up and going with the finished version of the URDF accessible on the ROS textbook's github. (contact a friend to learn to use github)

- Git clone https://github.com/PacktPublishing/ROS-Robotics-Projects-SecondEdition
- For your info: Urdf and other file formats found here: **https://github.com/PacktPublishing/ROSRobotics-Projects-SecondEdition/chapter_3_ws/src/ robot_description/**

This workspace (chapter_3_ws) needs to be compiled as a ROS package.

MOVING THE CAR WITH A JOINT_CONTROLLER

Follow instructions from the ROS Textbook:

- **Testing the robot base**
- **PG100 (stop at "Modeling the robot arm")**

SETTING UP POSTOFFICE WORLD

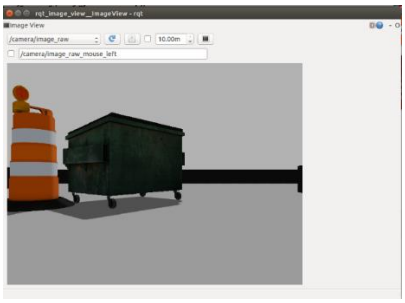
A template world can be loaded from the gazebo libraries.

Follow instructions from the ROS Textbook:

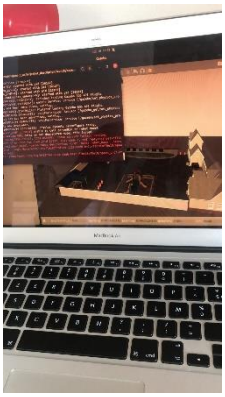
- **Setting up the environment in Gazebo**
- **PG168 (stop at "Making our robot base intelligent")**
- **Be careful to spawn the robot base and not mobile_manipulator as they do. You can compile chapter_5_ws as they do, or keep the previous workspace.**

ADDING A CUSTOM CAMERA TO THE BOT

Gazebo allows the integration of cameras in order to visualise the simulation environment.



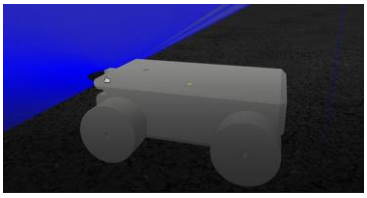
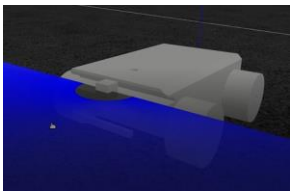
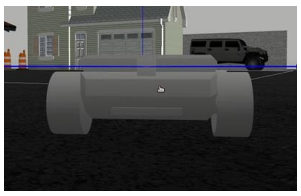
I followed a tutorial provided by Gazebo documentation in order to integrate a custom camera (top). Read it but do it at your own risk.



Error 1: Main difficulty: the tutorial establishes a child-parent link between the camera (child) and some other link, which in our case should be base_link (parent). As a result: parse errors ; the model will not be able to spawn. See my github model for those lines… or just figure out how to copy my model's code.

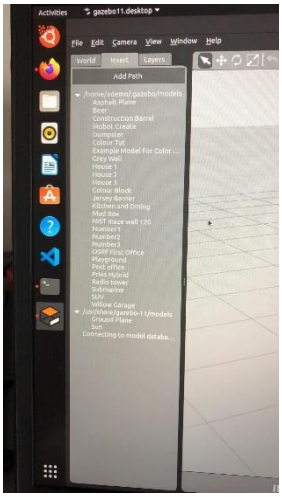
Error 2: The gazebo thread might register a failed camera spawn. Make sure you know how to view rviz output of the camera view on the robot. The camera should still appear, making the car fully functional.

Do similarly to integrate a custom lidar (bottom).

ROS Textbook - https://drive.google.com/file/d/147l-EPXTe4QW_H9m5FHB2yH3JsMd5oMQ/view?usp=sharing (request access first)

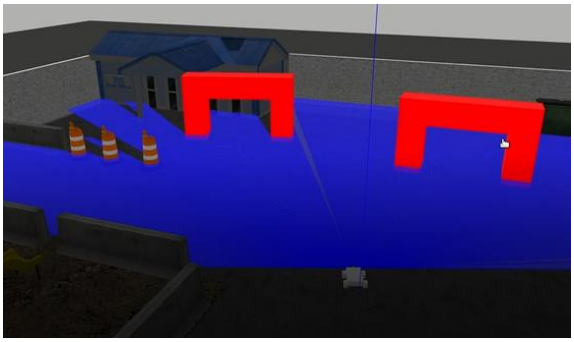
MAKING A RED GATE

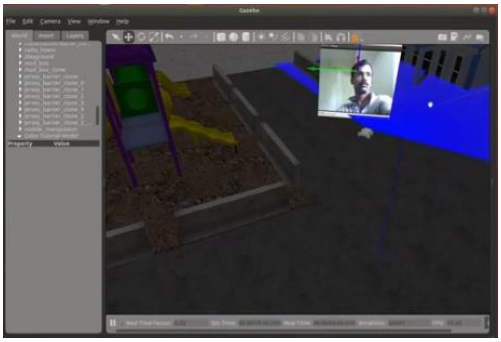


In order to make the waypoint itself, it was important to examine how textures and colours are created in the Gazebo world, which I discovered in this tutorial. Once again: read it but don't do it yourself 😊 because there are too many mistakes in the tutorial! The major difficulty: Adding colour textures, but also custom images, onto an object in Gazebo, actually requires the addition of a separate package for texturing.

On the github, the folder **models_for_gazebo** have 3 models. It seemed simplest for me to fetch the model from any simulator instance by storing it in the full object in the root /.gazebo/models folder. Do this with the models linked above.

- Gate
- Indian block
- Example model → Appears as "colour_tut" in tab alongside.





DETECTING OBSTACLES

For detecting obstacles, I used some basic OpenCV.

OpenCV - recognise colours and calculate position on image

Watch out: As of May 2020, there was no support for OpenCV6 in the ROS libraries specific to Python3, therefore a ROS node using OpenCV6 and python2 was put together using various resources.

For now: ask txa for his package which you will be able to modify!!

- FIRST TEST WITH YOUR WEBCAM: https://github.com/ThomasCarstens/cours-de-robotique/blob/main/webcam_door.py (some configuration might be required.)
    - Make sure to use python2: syntax errors might occur due to incorrect tabs.

- OPENCV SCRIPT ANNOTATED: https://github.com/ThomasCarstens/cours-de-robotique/blob/main/cvbridge_v1.4.py

This code will not work at first. You will be using this script as a **template** for the next stage. There is no ROS connection, and connecting a ROS node (as input and output images) will tell if your script is working!

```
58      hsv = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV) #Camera video to HSV
59      redmask = cv2.inRange(hsv, l_b2, u_b2)
60      #-----Reduce Noise using morphology opening-----
61      R_opening_masked_img = cv2.morphologyEx(redmask, cv2.MORPH_OPEN, kernel)
62      #-----Resulting image-----
63      R_masked_img = cv2.bitwise_and(cv_image, cv_image, mask=redmask)
64      #-----Finding contours and drawing it on the result image-----
65      (R_contours, R_hierarchy) = cv2.findContours(R_opening_masked_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
66      #cv2.drawContours(masked_img, contours, -1, (255,255,255), 3)
67
68      ###################### PUTTING INFO ON THE CAMERA PICTURE ######################
69      cv2.line(cv_image,(400,480),(x2,y2),(0,255,0),1)
70      font = cv2.FONT_HERSHEY_COMPLEX
71      if -velocity<0 : #adding text overlay!
72      text = 'Gauche'
73      else :
74      text = 'Droite'
75      cv2.putText(cv_image, text, (20, 20), font, 1, (0, 255, 0), 1, cv2.LINE_4)
76      cv2.rectangle(cv_image, (x, y), (x + w, y + h), (0, 255, 0), 2)
77
78      cv2.drawContours(frame, contours, -1, (0,255,0), 2)#dessiner contours
79
80  ################################## OBSTACLE EXISTENCE ##############################
81      cv2.imshow("Green_maskimg", G_masked_img)
82      cv2.waitKey(1)
```

- More details about this code can be found in the API https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html
- There are tutorials to learn OpenCV, but it's not required for this project and:
    - you need to dive pretty deep to understand morphological transforms
    - it's not useful for this project nor gesture recognition because the Haar Cascades do not use OpenCV.

RELATED (NOT PART OF PROJECT): USING ROS WITH A MACHINE LEARNING ALGORITHM

Haar Cascades – face recognition or gesture recognition

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. A little video shows the car approaching the face. (**at 1:05)** The rest of the video shows the car setup.

https://www.youtube.com/watch?v=tNqYDqC6wo4&t=65s&ab_channel=ThomasCarstens

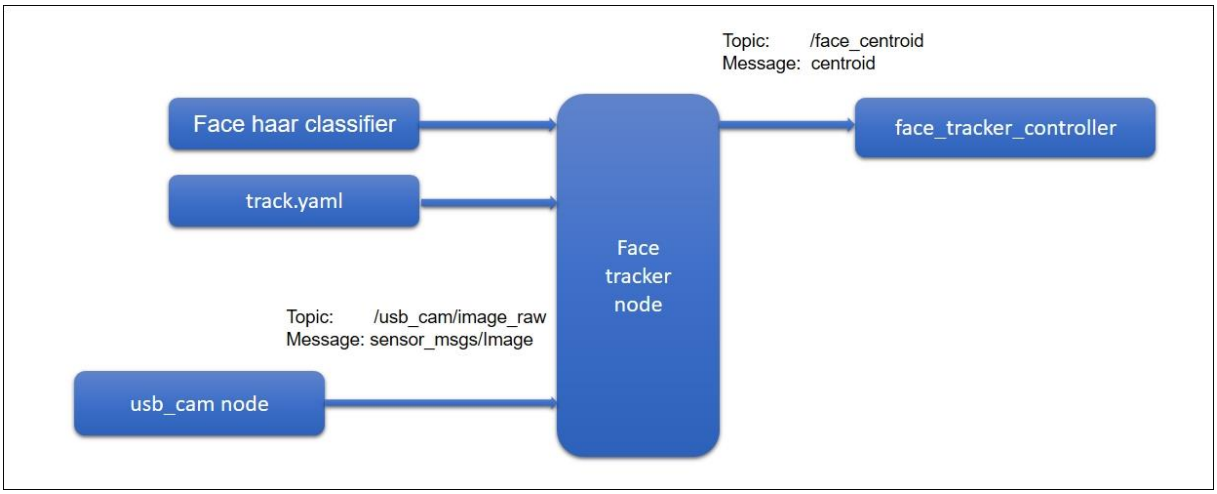
This was done with a **haar cascade.** It is an algorithm that learns to distinguish overarching pixel patterns in an image. Quick overview: https://www.youtube.com/watch?v=jG3bu0tjFbk

For building a haar cascade for other inputs (hand gestures, etc), refer to this series of videos.

- **CUSTOM**
- **Making your own Haar Cascade Intro - OpenCV with Python for Image and Video Analysis 17 (See numbers 16 and 17**
- https://www.youtube.com/watch?v=jG3bu0tjFbk

USING HAAR CASCADES INSIDE ROS

The ROS package for Haar cascades have a particular architecture which would be useful to learn about, on pg397 of the ROS manual.



I replicated a Haar cascade as shown in the ROS Textbook.

STAGE 2: SETTING UP ROS

You now should have:

- control over the robot's motion
- The ability to detect the gate via its colour and dimensions

This part is about interlinking the input to the output as simply as you can. 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 will call this 'event-based' programming.**

MOVING THE CAR USING A ROS PUBLISHER

Start from scratch again. Write a Python script that publishes the car's motion. Perhaps get up and going the the ROS Tutorials for ROS Publishers using ROS; then finding the correct topic to publish to, and figuring out in which format to do so.

WHEELED BOT NAVIGATION FROM IMAGE TO WHEELS

Different behaviours had to be coded according to what was detected in the images.

Look at the template that I provided: https://github.com/ThomasCarstens/cours-de-robotique/blob/main/cvbridge_v1.4.py

The required nodes are labelled by comments.
Start by inserting the required ROS nodes into the script. Test gradually. You do not need to put the python scripts in your workspace in order for them to work. For the sake of neatness, we conventionally store them in ~/catkin_ws/src/scripts/

FINETUNING THE ROBOT MOTION

This is an experimentation stage.

The robot motion required different speeds for the different parts of the trajectory. Only with specific tuning could I achieve a bot driving toward the gate.

DETECT THE SIDES OF THE GATE WITH A LIDAR

In the last part, the car must go inside the gate: the lidar is there to detect the sides of the gate and avoid it.

- First learn to use the lidar: see code-functionality.py for hints.
- Then fuse camera and lidar information in the final script.

STAGE 3: SETTING IT UP FOR A DEMO

LAUNCH FILE FOR ALL SCRIPTS AT ONCE + COMMON LOG FOR ALL SCRIPTS

Learning how to launch different nodes for project launch.
http://wiki.ros.org/ROS/Tutorials/UsingRqtconsoleRoslaunch

RELOADING THE CAR AS SOON AS IT COLLIDES WITH AN OBJECT

A recap of service calls will be useful.

http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

This requires collision detection.

https://github.com/SteveMacenski/gazebo_model_collision_plugin

For respawning the robot, learn how to use Gazebo service calls explained below.

http://gazebosim.org/tutorials/?tut=ros_comm

HOW TO STOP THE PYTHON SCRIPT: WRAPPING CODE UP IN AN ACTION SERVER

We're on the last stage! Finally, a central decision planner is used in preparation for scaling up the network. To do this, a ROS Action Server wraps the actions. Using a mission_state ROS node, you can run a full Python Server that in itself runs **an action** in its callback.

First make sure you understand how to perform an **action.** Follow this tutorial. You can later adapt it to your specific action.

http://wiki.ros.org/actionlib

Using this server, you can create a custom message within a client that returns the amount of time it takes for the robot before a collision.

https://index.ros.org/doc/ros2/Tutorials/Actions/Writing-an-Action-Server-Python/

Custom actions can deliver specific information to the user once the action has completed!

GOING FURTHER (NOT PART OF PROJECT): USING A STATE MACHINE TO CONTROL THE SEQUENCE

Splitting the robot motion into states is a technique to be able to create various robot applications by **revisiting** states at will.

We make use of a state machine to be our 'decision-maker' (state transitions) between the sensor input (callbacks) and specific motions of the bot (actions).

ROS Textbook - https://drive.google.com/file/d/147l-EPXTe4QW_H9m5FHB2yH3JsMd5oMQ/view?usp=sharing (request access first)

SMACH is a simple Python implementation of state machines, and for the purposes of developing various usecases, you can get up and running easily.

This is the subject of a separate tutorial.