FINAL PROJECT

Python Applications for Robotics

May 7, 2021

Instructors:
Z. Kootbally

Group:
Group Number 11

Semester:
Spring 2021

Course code:
ENPM809E

Students:
Paras Savnani

Santosh Kesani

Gautam Vanama

Wenliang Zhen

Contents

1 Introduction

The essence of the project is to utilize a technique known as Visual Servoing to control the motion of a robot. Visual Servoing is the method of controlling a robot's motion using real-time feedback from vision sensors to execute tasks like tracking objects and following objects.

For this project, we drive two turtlebots in a house environment(simulated in Gazebo) to move between predetermined way-points. Among these two bots, one drives around using mapping and localization where as the other has to utilize visual servoing to follow the first one. The one that uses mapping and localization is named as leader and the other one is follower.

In order to utilize the capabilities of visual servoing technique in gazebo environment, the leader has been provided with an Aruco Marker, which will be perceived by the follower's camera to follow the leader and also enables the follower to keep a safe distance between the two of them.

2 Approach

- **Controlling Leader**

  The leader robot is fed with predetermined way points in the environment and it performs autonomous navigation to move from one preset location to the other. So, as for any autonomous navigation, it first requires a map and in this case has been generated using the slam_gmapping node from the gmapping package. Next step was to access this map which was done using the map_server node from map_server package. Later on to perform the required path planning to successfully move from one point to the other within a map the leader uses the move_base node from the move_base package. Finally, to continuously localize the robot in the map it utilizes uses the amcl node from the amcl package and the benefit of this is package is the more the robot moves in the environment, the more data the robot will get from sensor readings, which results in more accurate localization. This approach ensures that the robot is capable of performing navigation while avoiding the obstacles in the environment.

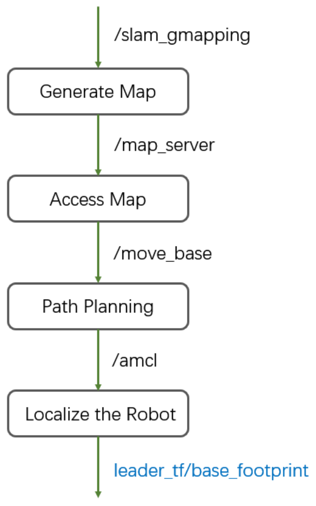


Figure 1: Controlling Leader

```
# 1) Read Yaml file consisting of location coordinates
        - Store them in a dictionary with the key name pointing
          toward the location and with the room coordinates as the corresponding values.
        locations = OrderedDict()
        with open(path, 'r') as file:
            yaml_info = yaml.load(file)
            for key, value in yaml_info.items():
                locations[key].append(value[0]['position'][0]['x'])
                locations[key].append(value[0]['position'][0]['y'])
                locations[key].append(value[0]['orientation'][0]['w'])

# 2) Now that we have our locations, we use MoveBaseGoal to set the target
#    positions to the robot
        goal = MoveBaseGoal()
        goal.target_pose.header.frame_id = "map"
        goal.target_pose.header.stamp = rospy.Time.now()
        goal.target_pose.pose.position.x = float(coordinates[0])
        goal.target_pose.pose.position.y = float(coordinates[1])
        goal.target_pose.pose.orientation.w = float(coordinates[2])

        client.send_goal(goal)
        wait = client.wait_for_result()
```

- **Pose Broadcaster**

  This node is responsible for initializing a subscriber to recieve fiducial transforms from the aruco marker detection. The transformation between the map and the marker is not directly obtained by can be found by determining the the following three transformations: **map-robot, robot-camera, camera-marker**. The fiducial transforms contain the pose of the marker with respect to the follower camera frame. We create a tf broadcaster to transmit these transforms to the tf tree an finally we perform a lookup transform to get the pose of the marker with respect to the map frame.

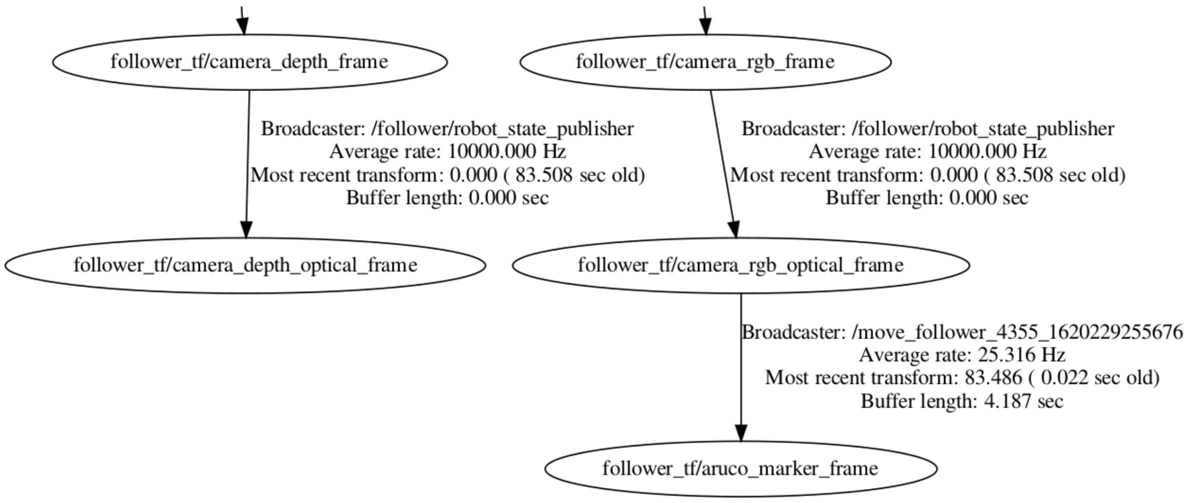

follower_tf/camera_depth_frame
follower_tf/camera_rgb_frame
Broadcaster: /follower/robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (83.508 sec old)
Buffer length: 0.000 sec
Broadcaster: /follower/robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (83.508 sec old)
Buffer length: 0.000 sec
follower_tf/camera_depth_optical_frame
follower_tf/camera_rgb_optical_frame
Broadcaster: /move_follower_4355_1620229255676
Average rate: 25.316 Hz
Most recent transform: 83.486 (0.022 sec old)
Buffer length: 4.187 sec
follower_tf/aruco_marker_frame

Figure 2: Pose Broadcaster

```
#
# 1) set transform listener
listener = tf.TransformListener()

# 2) Subscribe to incoming fiducial_transforms messages
rospy.Subscriber("/fiducial_transforms", FiducialTransformArray, callback)

# 3) Publish transform to each fiducial
br = tf.TransformBroadcaster()

# callback function is called whenever our subscriber
# receives fiducial transformations.
def callback(msg):
    # For every fiducial found, publish a transform
    for m in msg.transforms:
        id = m.fiducial_id
        trans = m.transform.translation
        rot = m.transform.rotation
        br.sendTransform((trans.x, trans.y, trans.z),
                         (rot.x, rot.y, rot.z, rot.w), rospy.Time.now(),
                         "follower_tf/aruco_marker_frame",
                         "follower_tf/camera_rgb_optical_frame")
```

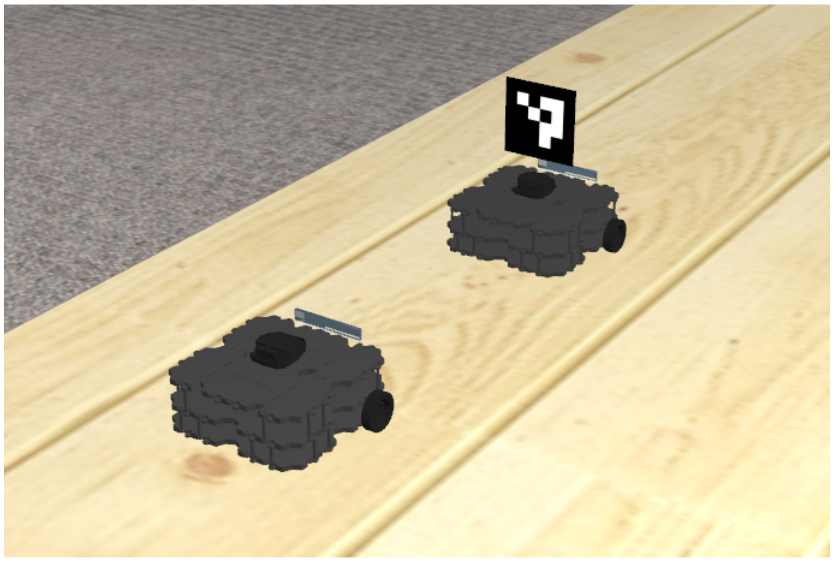


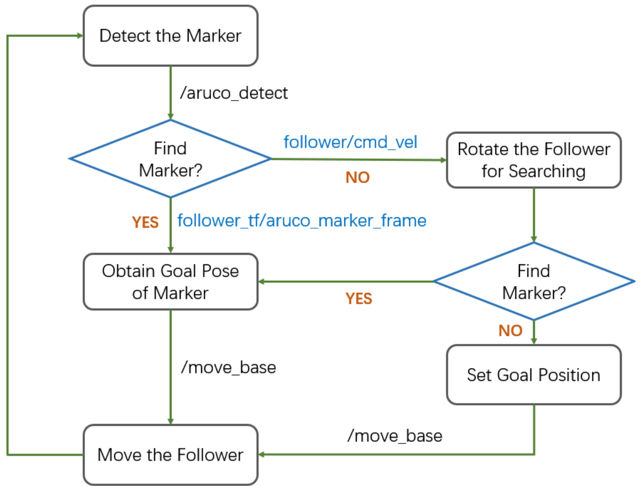
Figure 3: Pose Broadcaster

- **Controlling Follower**

  The follower's task is to detect the marker placed on top of the leader and use the move_base action client to move to the marker's location. In order to achieve this we would need the location of the marker in global co-ordinates which can be obtained by the transformation between the map and the marker using the pose broadcaster node and lookup transformation. So, by keeping a track of these transformations we continuously obtain the marker's location and drive the follower. move_base client is a recursive function which calls itself continuously till the marker is in the sight and waits for the server to execute the maneuvers. It stores the transforms into a buffer variable to enable the traversal to the latest point the marker was detected and it continuously updates this variable.

If the marker is lost from the camera sight of the follower camera, the lookup transforms through an error and the search protocol is executed in the except block. Following the protocol, the follower first rotates for 360 degrees and if it finds the marker it follows it, else it reads the parameter server value, set by the leader and tries to reach that location using move_base to increase the probability of detection. If it reaches the position and still doesn't find the marker, it rotates continuously at that position.

The challenging part in this section is to make the follower continuously follow the marker without loosing its sight and to take into account the rotations and the translations of the leader to calculate relative angular error rates and synchronize the speed.



```
# 1) Get the fiducial position - rotation and translation
(trans,rot) = listener.lookupTransform('/map',
                                        '/follower_tf/aruco_marker_frame',
                                        rospy.Time(0))]
alpha = 0.9
rotation0 = quaternion_to_numpy(rot)
rotation = quaternion_to_numpy(rot)
rotation_interpolated = quaternion_slerp(rotation0, rotation, 1 - alpha)

translation0 = translation_to_numpy(trans)
translation = translation_to_numpy(trans)
translation = alpha * translation0 + (1 - alpha) * translation

# 2) Update your robot position to the fiducial by calling movebase_client
#    by sending the updated positions
    latest_pos = [translation[0] - tolerance, translation[1] -
                  tolerance, rotation_interpolated[3]]
    result = movebase_client(latest_pos)

# 3) movebase_client function helps the robot navigate to the goal point
    if the marker is found print "marker found"
```

```
def movebase_client(coordinates):
    while(1):
        wait = client.wait_for_result(rospy.Duration(0.2))
         # Check if the robot has reached the goal
        if wait:
            break
        try:
            (trans,rot) = listener.lookupTransform('/map',
                                '/follower_tf/aruco_marker_frame', rospy.Time(0))
            rotation_interpolated = quaternion_slerp(rotation0, rotation, 1 - alpha)
            translation = alpha * translation0 + (1 - alpha) * translation

            latest_pos = [translation[0] - tolerance, translation[1]
                            - tolerance, rotation_interpolated[3]]

            client.cancel_goal()

            if not last_print == "Marker Found \nFollowing the marker":
                last_print = "Marker Found \nFollowing the marker"
            else:
                last_print = "Marker Found \nFollowing the marker"
            result = movebase_client(latest_pos)
            return None

        except:
            # Check if the robot is already at the goal
            (trans1,rot1) = listener.lookupTransform('/map',
                        '/follower_tf/camera_rgb_optical_frame',
                        rospy.Time(0))
            if (abs(trans1[0] - coordinates[0]) < 0.10) or
                (abs(trans1[1] - coordinates[1]) < 0.10):
                current_goal = rospy.get_param("current_goal")
                if current_goal[0] == "bedroom":
                    if (abs(trans1[0] -
                        (current_goal[1][0]+ 0.5))] < 0.15)
                        and (abs(trans1[1] -
                        (current_goal[1][1] + 0.5)) < 0.15):
                        exec_flag = True
                return None

# 4) If the fiducial marker is not found rotate and search for the marker
velocity_msg.angular.z = -3.0
pub.publish(velocity_msg)

# 5) If the fudicial marker is not found even after scanning around
direct the follower robot to move toward the leader robot goal position
result = movebase_client([current_goal[1][0] + 0.5,
        current_goal[1][1] + 0.5, current_goal[1][2]])
```

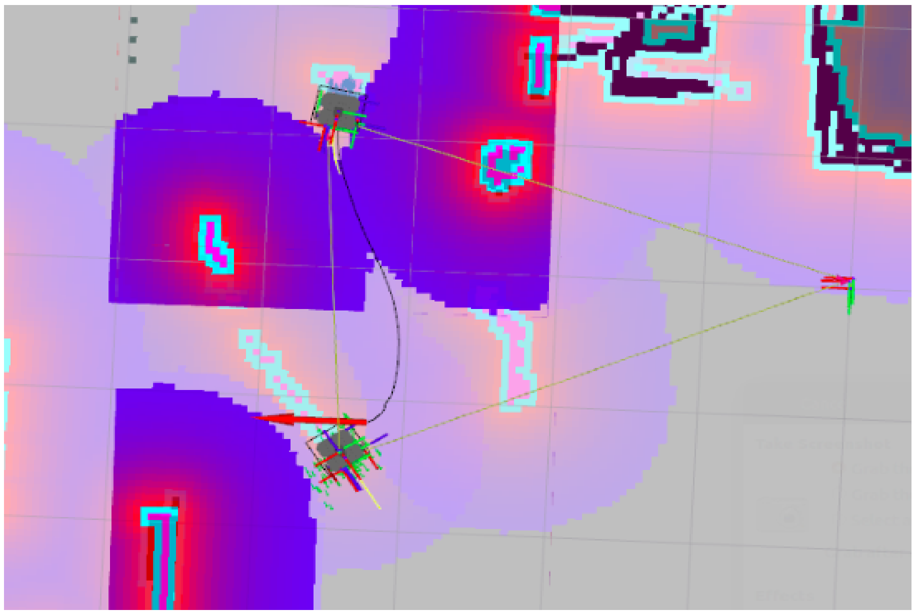


Figure 4: Follower following the leader robot after detecting the aruco marker

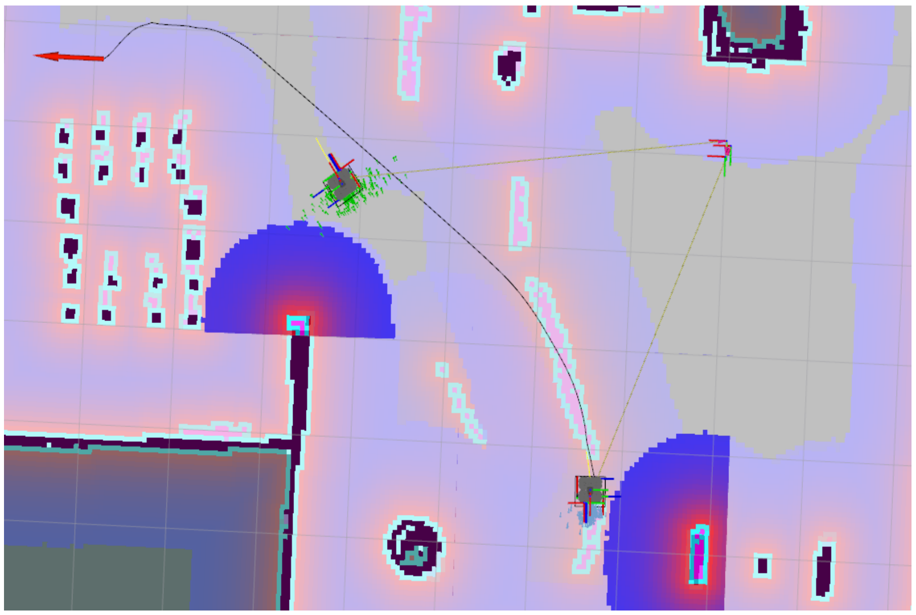


Figure 5: Follower heading towards the parameter server

3 Challenges

- For the move_follower node, move_base tends to follow the optimal path to reach the location so sometimes it tends to move in the reverse direction showing the blank side of the aruco marker, so it was impossible to detect the marker even if the follower was right behind the leader.(Possible solution tried: Attached the marker on both sides to improve the visual servoing.)
- We can only receive transformations if the marker is in the frame, so we had to maintain a "buffer variable" to keep a record of the latest transformations if they are not available and allow continuous movement of the robot.
- Sometimes, in exploration phase follower robot reached the parameter server position before the leader robot and leader was unable to reach the position. So, we had to offset the follower position by 0.5 units so prevent the leader from getting stuck.
- While following the leader, if the follower speed is not synchronized it tends to collide with the leader, so the follower was tasked to keep a safe distance (0.2 units) from the marker to avoid collision.
- The follower robot got stuck sometimes while following the marker, so the move_base client function was made recursive with a threshold breaking condition to allow the robot to enter the exploration phase of rotation and navigating to the parameter server.
- Rotation speeds had to be tweaked to allow faster detection of the marker, else the leader robot got deviated from the sight.
- Fiducial transforms were not accurate, so we changed the translation and quaternion coordinates to numpy array and applied an averaging factor alpha with a value of 0.9 to improve the transformation accuracy.
- To increase the robustness of the code multiple spawning positions were tried and tested in the environment.
- Multiple parameters in the param files (global cost map and local cost map etc.) had to be tweaked to resolve the recovery behaviour of the move_base and remove frequent warnings from the cost map deprecations.
- For the pose_broadcaster node, using the tf.Transformbroadcaster() in the callback function to send the fiducial transforms to the aruco marker frame attached with the follower camera frame.

4 Project Contribution

- **Santosh** - I worked on reading the location_recorder yaml file with locations and driving the leader to way-points in the yaml file. Also, helped in finding resources for visual servoing.
- **Paras** - I improved the leader driving part of the code to make it robust and readable. Furthermore, I implemented the pose broadcaster node to broadcast the fiducial transformations to the follower camera frame and consecutively do a lookup transform to get the pose of the marker. Also, wrote the move_follower node using the move_base client to track and follow the leader robot and use the search protocol when marker is not in sight of the camera.

- **Gautam** - Most of my contribution was devoted to figuring out on how to make the follower robot follow the leader as close as possible. Tried and tested few approaches to get the desired result we were looking for. We all worked on getting the leader robot traverse across all the locations and chose the most readable and optimized solution.
- **Wenliang** - I managed to let the leader navigate in the environment according to the location.file. Try to make the follower track the marker movement, but failed. With the help of my teammates, I understood how to make the follower do visual servoing, and provided suggestions for achieving faster marker detection.

5 Resources

- map_server package
- move_base package
- actionlib
- amcl package
- navigation
- aruco_detect package
- tf_broadcaster
- Fiducials

6 Course Feedback

- **Santosh** - Improved my python skills a lot and learned a few unfamiliar python related concepts which I think will be helpful in my future projects. Using ROS with python was a new experience and helped me realize the flexibility that I can have in developing with python. The projects have allowed me to learn new concepts in ROS like mapping and localization. Also, learning about the visual servoing technique was quite interesting as it also used in real-time industrial robots. Testing and debugging the final project was a little troublesome due to the hardware limitations of my laptop.
- **Paras** - I was already familiar with python, but this course did help me to level up my python skills. I learned about ROS and its basic features like publisher, subscriber, services, messages etc. I think more time should be devoted to ROS(6-8 weeks) rather then python basics to make it more interesting. Apart from it, I found assignments to be quite easy, so if assignments could be made more challenging, it would improve the overall experience.
- **Gautam** - The course was good overall, I got to refresh my memory on Python, and I would recommend this course to anyone who is not familiar with python as the Professor teaches the important concepts and covers a fair bit of background, but I felt that more emphasis on ROS than on python would have given us time to get ourselves hands on with ROS. That said, it was a good introduction to Robot Operating System as well.
- **Wenliang** - The biggest gain for me from this course is how to use Publisher/Subscriber and Client/Server to control the robot. Also improved my python programming skills. This course has a very good explanation on how to use ROS. The navigation algorithm based on the target position and the closed-loop speed control algorithm are very helpful to my work, and it is a very good study case. The most difficult part for me is Python programming to implement a

complete motion control system. My programming ability is not enough to execute my ideas completely. I think RWA-2 and the final project are more difficult for me, because I have basically never learned python. The improvement I hope to see is that there are more cases to learn from. In my opinion, there is a big gap between the difficulty of the class case and the assignments.

7 References

- Nick Lamprianidis (2020). *Visual Servoing in Gazebo*
- pal-robotics. *aruco_ros Github Repository*
- How to write a broadcaster using tf in ROS. *ROS wiki*
- How to Apply running average filter to translation and rotation. *Github Repository*