Documentation for solutions of 'Detecting advanced lane features ROS-test'

Sandeep Mattepu, Tony John

Contents

1 Task 1 : Read and obey the README.md(Remove all errors from the package and atleast two launch files has to work) 3
- 1.1 Error 1 : "Could not find package joy" produced by CMakeLists.txt file when compiling the package . . . 3
  - 1.1.1 Solution : . . . 3
  - 1.1.2 Approach : . . . 3
- 1.2 Error 2 : "Has no member named 'turn' in inf_main.cpp" and "Has no member named 'forward' in inf_main.cpp" produced when compiling the package . . . 3
  - 1.2.1 Solution : . . . 3
  - 1.2.2 Approach : . . . 3
- 1.3 Error 3 : "Requires the 'velocities' arg to be set" produced when trying to launch inf.launch file . . . 4
  - 1.3.1 Solution : . . . 4
  - 1.3.2 Approach : . . . 4
- 1.4 Error 4 : Turtle doesn't go in square as code written in main.cpp when ros_test.launch file is launched . . . 4
  - 1.4.1 Solution : . . . 4
  - 1.4.2 Approach : . . . 4

2 Task 2 : Create a node which draws the/an infinity sign using the "arc"-function 6
- 2.1 Solution : . . . 6
- 2.2 Approach : . . . 6

3 Task 3 : Create a ROS-msg with the status values of the turtle and send it frequently **7**
3.1 Solution : 7
3.2 Approach : 7

4 Task 4 : Create an node for drawing a binary tree with the turtle. **8**
4.1 Solution : 8
4.2 Approach : 8

1 Task 1 : Read and obey the README.md(Remove all errors from the package and atleast two launch files has to work)

1.1 Error 1 : "Could not find package joy" produced by CMakeLists.txt file when compiling the package

1.1.1 Solution :

Remove package name **joy** from *find_package* block in CMakeLists.txt file.

1.1.2 Approach :

1. Tried to compile package using *catkin_make* and it produced error in terminal stating the error.
2. Looked at all the source code files if any part of the code is using package called **joy** and we found none.
3. We removed the **joy** package and recompiled. The package missing error is fixed.

1.2 Error 2 : "Has no member named 'turn' in inf_main.cpp" and "Has no member named 'forward' in inf_main.cpp" produced when compiling the package

1.2.1 Solution :

Change all the lines in inf_main.cpp file where members of *turtle* variable are being accessed with dot operator(Eg:- *turtle.forward(3)*), replace them with arrow operator(Eg:- *turtle->forward(3)*)

1.2.2 Approach :

1. Tried to compile package using *catkin_make* and it produced errors in the terminal stating that problem is from inf_main.cpp file.
2. Looked at data type of *turtle* variable which is *std::shared_ptr<>* type.

3. Replaced all the dot operators with arrow operators.

4. Recompiled the project and this error got fixed.

1.3 Error 3 : "Requires the 'velocities' arg to be set" produced when trying to launch inf.launch file

1.3.1 Solution :

Change the line *<param name="velocity" value="$(arg velocities)" />* to *<param name="velocity" value="$(arg velocity)" />* in inf.launch file

1.3.2 Approach :

1. Tried to launch inf.launch in terminal but it shows error in the terminal stating that it requires the *velocities* arg to be set.

2. By fixing the typo from *velocities* to *velocity* in inf.launch file the error is fixed.

1.4 Error 4 : Turtle doesn't go in square as code written in main.cpp when ros_test.launch file is launched

1.4.1 Solution :

In turtule_abstract.h file, inside the constructor of the *AbstractTurtle* class, change the topic name that is being passed to *nh.advertise( )* function from *"turtle1_cmd_vel"* to *"turtle1/cmd_vel"*.

1.4.2 Approach :

1. Checked for any logical errors in main.cpp and turtle_abstract.h files.

2. Launched the ros_test.launch file and opened seperate terminal for debugging using *rostopic* and *rosnode*.

3. By typing following commands in terminal, we were able to see what topics that */turtle_controll* was publishing and what topics that */turtlesim* was subscribed to.

```
rosnode info /turtle_controll
rosnode info /turtlesim
```

4. There was a mismatch in the name of the topic for publisher(*turtle_controll*) and subscriber(*turtlesim*). Fixing the name in the code has made the turtle draw a square.

2 Task 2 : Create a node which draws the/an infinity sign using the "arc"-function[1]

2.1 Solution :

1. Pass the *use_arc* flag to *draw_u* function inside inf_main.cpp file.

2. If it is true, use *arc* function for drawing the 'U' with a radius of 1.5 and angle of 180.

```
if(use_arc)
{
    turtle->arc(sign*1.5, 180);
    turtle->turn(sign*45);
}
```

2.2 Approach :

1. Looked into inf_main.cpp file. Found that it takes the *use_arc* flag but is not used.

2. Looked into turtle_abstract.h file. Found out that *arc* function takes *radius* and *angle* of arc as arguments. It could be used in the *draw_u* function in inf_main.cpp.

[1]Please note that after looking at the code inside inf_main.cpp file we thought that solving this task is just about making this code work properly(print infinity sign when inf.launch file is launched). The explanation here shows how we did it. If your intention was to make us create a new node from scratch and make it print infinity sign, we did this as well you need to checkout *arc-function-node* branch and launch **inf_arc.launch**. Description of this task has made us confuse

3 Task 3 : Create a ROS-msg with the status values of the turtle and send it frequently

3.1 Solution :

1. Create a msg file - msg/TurtleStatus.msg
2. Turtlesim/Pose message has the position and velocity parameters so we decided to use that in the TurtleStatus.msg, and additional parameters for accelerations and total distance travelled were added as follows.

```
turtlesim/Pose position_velocity
float64 linear_acceleration
float64 angular_acceleration
float64 distance_walked
```

3. Made necessary changes to CMakeLists.txt and package.xml for message generation.
4. Created a publisher for TurtleStatus msg on the topic */ros_test/turtle_status*.
5. Calculate the parameters using the current position from the *Pose* message and publish the turtle_status message in the *poseCallback* function.

3.2 Approach :

1. Create a custom message including the required parameters.
2. Add this to CMakeLists.txt.
3. Make necessary changes to package.xml for message generation.
4. Create a topic and publish the message with updated parameters whenever the turtle moves.

4 Task 4 : Create an node for drawing a binary tree with the turtle

4.1 Solution :

To see the solution, launch it by typing the following command.

```
roslaunch ros_test draw_binary_tree.launch
 velocity:="1.0" length:="3.8" angle:="45"
 factor:="0.5" depth:="3" branches:="3"
```

4.2 Approach :

1. Created *BinaryTreeDrawer* class which utilizes *tutorial::AbstractTurtle* instance to draw binary tree.

2. The *drawBranches( )* function in *BinaryTreeDrawer* contains the logic how it guides the turtle to draw a binary tree. It is a recursive function. Each recursion ends until it draws the line in the deepest branch.

3. The *collision_aware_forward( )* function is added to *tutorial::AbstractTurtle* class which performs similar to *forward( )* function already present in the class. This function stops forward motion when turtle is closer to turtlesim boundaries. With the help of this function *BinaryTreeDrawer* can draw branches which are in bounds of the turtlesim or else it will skip drawing the current branch and starts drawing other branch.

4. *BinaryTreeDrawer* class is used in draw_binary_tree.cpp for creating a node that actually sends instructions to turtlesim node. CMake-Lists.txt and package.xml files are also updated to create a executable.

5. Created a new launch file that can be launched which is discussed in previous section.