ESA-ROS Assignment 2

Minh-Triet Diep, Lars Jaeqx

Code explanation

Point and shoot

We first started with calculating the angle we need to turn to face the coordinates we want to travel to. This happens in the cbGoal callback. We have our own position and the goal position, and with some goniometry we can calculate the heading we need. This boils down to angle = atan(dy, dx), with dy and dx being the distance between the robot and the goal we get from the message.

We also have the robot's existing rotation, so we need to figure out what to turn to reach that angle we calculated earlier. In the first versions of the program, we simply substracted the angles to get the final rotation, and determined which direction to turn if the difference between them is positive or negative. This proved to make it turn unnecessarily much, so we figured out we needed to get the smallest angular distance between two angles. We used a function from Stackoverflow before we discovered this was something in the ROS library, so that code is still used.

With that, we managed to get it to turn to the goal with the smallest rotation needed. After that, we need to do the "shoot" part. This is relatively simple, as we use Pythagoras' theorem to get the distance. With this, we managed to get point-and-shoot working. The final part, rotating with the orientation part of the pose wasn't much trouble after applying what we had from the original rotation part. We did have some trouble making the orientation part of the message, so we used this on-line calculator to figure out the quaternion for the euler system

we know: http://www.andre-gaschler.com/rotationconverter/

This concludes "Point and shoot".

Servoing

We worked from the Point and Shoot and looked for how to make the rotate and shoot more accurate. We initially wanted to implement true pursuit, but after discussion found out this assignment didn't need that yet and only rotate and shoot needed to get things to be more accurate.

We got our PID implementation from a GitHub gist and used it in our application, as a library. This required some tweaking in the CMakeList.txt. For the PID values, we only chose to use the Proportional part. We tried to set the I and D values, but it didn't work and we didn't have enough background knowledge to implement the rest of that.

While making the rotate with the PID class, we found out that the controller didn't know the transition between -π and π, and always set the error in the direction of 0. We've tried to solve it with adding 2π to trick it, but that attempt didn't work, so right now the shortest angle isn't used. Inside the loop to rotate, we check our current angle with the angles from the odometry and we feed the PID controller with these values, to get the error.

While making the shoot function, we discovered that it overshot the target and continued. This is because the error we set was smaller than the drift it got from the rotation. We keep track of the error, and as soon as it increases again, we know we've overshot the target a tiny bit. This is where we stop. Again, as with rotation, we use odometry data to act upon.

Running instructions

As with the previous assignment, the steps are similar to get the program started:

```
cd ~/catkin_ws
catkin_make
source devel/setup.bash
roslaunch assignment2 assigment2_point.launch
roslaunch assignment2 assigment2_servoing.launch
```

To publish goals:

```
cd ~/catkin_ws
source devel/setup.bash
rostopic pub -1 /goal geometry_msgs/PoseStamped
'{header: {stamp: now, frame_id: "map"}, pose:
{position: {x: 1.0, y: 1.0, z: 0.0}, orientation: {x:
0.0, y: 0.0, z: 1.0, w: 0.0}}}'
```

Tests and Observations

We chose for YouTube videos instead of screencaps, because the trails weren't very clear.

Point and shoot

Results on YouTube: https://www.youtube.com/watch?v=CsxywmbSKs4

As you can see, the point-and-shoot method works reasonably well. Also visible is that once you need to go to the same coordinate you just went to, the robot does a small correction. This is because point-and-shoot has an inherent inaccuracy. We needed to use this rotation converter to figure out the rotation in quaternions, as is visible.

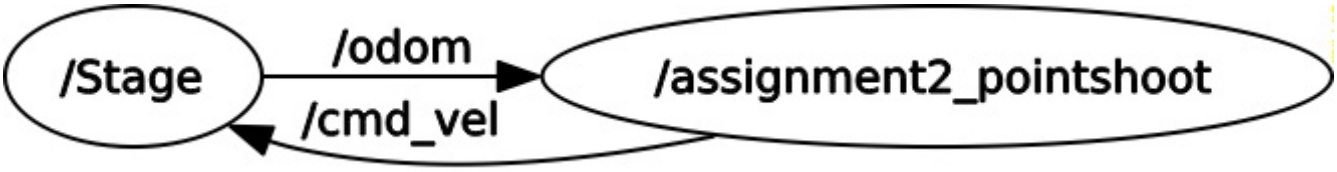
Servoing

Results on YouTube: https://www.youtube.com/watch?v=AumvYk4nCUU

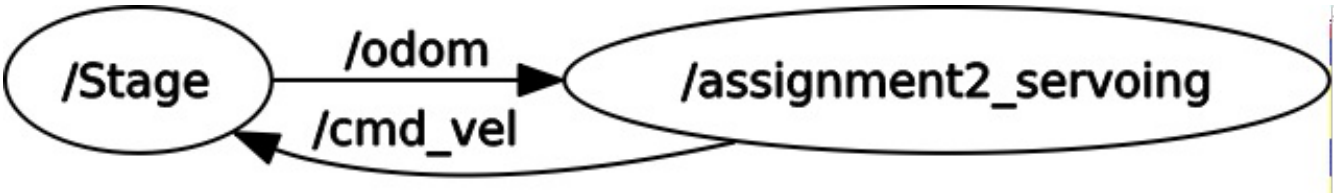
The described behavior of not turning over the -π and π degrees is visible here. As expected, the bot turns and moves, and when approaching the desired angles and coordinates, it slows down and stops.

Graph

Point and shoot



Servoing



We can see both graphs are similar. This is because we don't change any topics, and for the node we just changed the implementation.