Documentation

Bernhard Fritz, Mike Koch, Mario Zelger

Abstract

This document explains how to write IIS reports. It is written in the IIS Report style.

1 Introduction

Your document should begin with an introduction.

2 Technical Content

This is the bulk of your report, one or more sections, with nice illustrations. It is good style to cite people, not publications, as recommends Piater (2011). Web references should be avoided to the extent possible because they cannot usually be considered *archival* (Wikipedia), and require careful assessment of reliability.

2.1 Odometry

For odometry we implemented a simplified approach as an alternative to the technique discussed in lecture. In general, robot movement can be distinguished between rotational and translational movement. Our idea was that if you are aware of the robot's turn and movement speed, you have enough information to make it move wherever you want to. While turning/moving, the robot continuously updates its own estimated position and orientation. Of course due to the fixed time intervals there will be some small error over time but even the best odometry has troubles dealing with these kinds of problems.

2.2 Generating motion commands to attain goal location

After the first two exercises we concluded that we can no longer use the default moving and turning robot commands ('k' and 'l') since they were very inaccurate. So we decided to use a combination of the following commands instead:

- 'w' for moving forward

- 's' for stopping
- 'i' to set a specific velocity (we used this to turn in place)
- 'q' for sensor measurements

These commands as well as Java's capability of letting a thread sleep for a specific amount of time, enabled us to implement more precise motion commands. We decided to use a design pattern called command pattern as seen in figure 1 to structure our code. Following commands have been implemented:

- Translation
- GoTo
- RelativeRotation
- AbsoluteRotation
- Measurement

2.2.1 Translation

The *Translation* command can be considered as a command for relative robot movement. As soon as the *Translation* command is called we send a 'w' character to the robot. This results in sudden forward movement of the robot. Given a distance in cm as parameter and the robot's velocity we measured when we got the robot, we are able to calculate the time we need to wait to send a 's' character to the robot to make it stop. While the robot is moving we constantly keep track of its x and y coordinates in world coordinate system.

2.2.2 RelativeRotation

As soon as the *RelativeRotation* command is called we send an 'i' character with specific parameters to make the robot rotate in place counterclockwise. Given an angle in radiant as parameter and the robot's turn velocity we measured when we got the robot, we are able to calculate the time we need to wait to send a 's' character to the robot to make it stop. While the robot is turning we constantly keep track of its angle in world coordinate system.

2.2.3 Absolute Rotation

AbsoluteRotation is an extenstion of *RelativeRotation* and only uses some math to enable us to make the robot turn to a specific angle in world coordinate system.

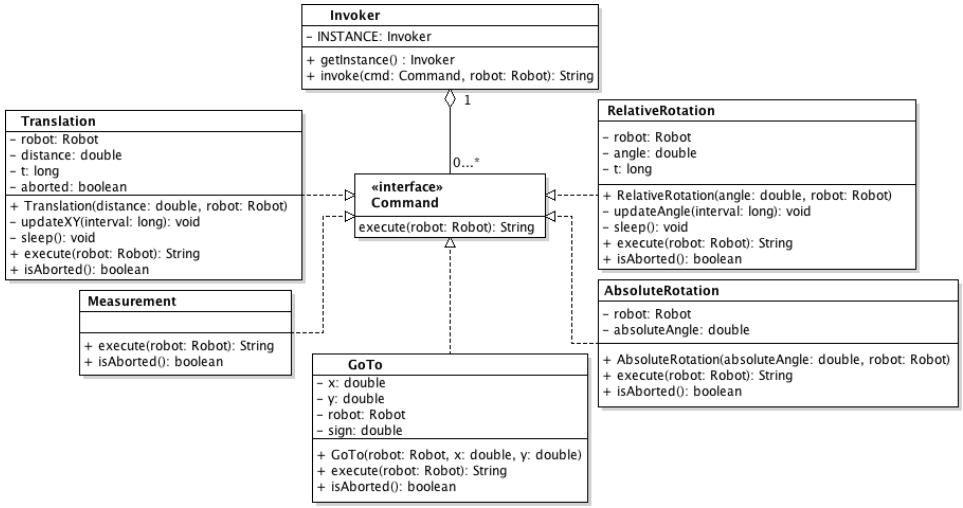

Figure 1: Command pattern

2.2.4 GoTo

GoTo consists of two commands: *AbsoluteRotation* and *Translation*. Given two parameters x and y (world coordinates) and the robot coordinates, we are able to calculate the angle we need to turn the robot so that it is facing the goal as well as the distance to the goal using trigonometry.

2.2.5 Measurement

The *Measurement* command is used whenever we want to read sensor values. We discovered that only three of five sensors are actually working:

- front left sensor
- front middle sensor
- front right sensor

The *Measurement* command is exclusively invoked by the *SensorManager* which is responsible for parsing received sensor data, calculating an average estimate of sensor values as well as notifying observers about imminent obstacles.

2.3 Obstacle avoidance

To avoid obstacles we needed a way to keep track of sensors while moving. At first we used a Java thread to move and poll the sensors at the same time. This didn't work too well since there should always be some delay between two robot commands. That is why we decided to try a different approach.

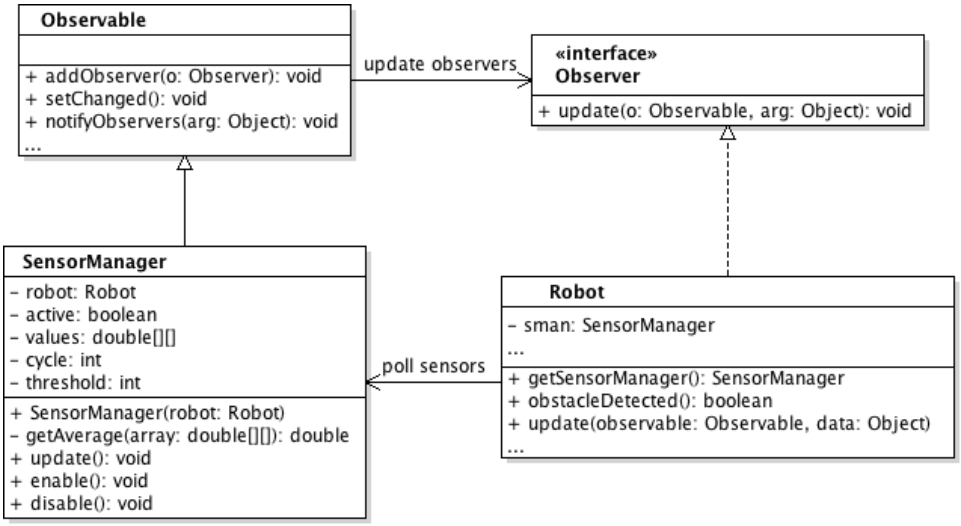

Figure 2: Observer pattern

Since it is not necessary to poll the sensors all the time (e.g. turning in place), we only focused on polling the sensors while moving. We used a design pattern called observer pattern as seen in figure 2 to conveniently let all observers know if an obstacle was detected or not.
If an obstacle was detected the current *Translation* command will be aborted and our obstacle avoidance algorithm will be started:

```
sign = 1;
while(Goal is not reached) {
  Turn towards goal;
  Move towards goal;
  if(Moving is aborted) {
    Turn sign*90°;
    Move 30 cm;
    if(Movement is aborted) {
      sign *= -1;
    }
  }
}
```

Essentially this algorithm is also known as "Bug 0" algorithm as seen in (Howie Choset, 2007, 7)

3 Conclusions

Your document should conclude with a summary of the highlights, and any conclusions to be drawn.

References

Howie Choset. Robotic Motion Planning - Bug Algorithms. `http://www.cs.cmu.edu/~motionplanning/lecture/Chap2-Bug-Alg_howie.pdf`, 2007.

Justus Piater. IIS report instructions. Explanatory notes, University of Innsbruck, 2011.

Wikipedia. Archive. `http://en.wikipedia.org/w/index.php?title=Archive&oldid=437062217`, 2011. Retrieved July 22, 2011.