MP Design

Oct 24, 2017

Yiran Zhao

Putting It All Together

- Covered so far:
  - System reliability
  - Data reliability
  - Timeliness

- Design problem: Design a “safe” robot

Requirement Engineering

- Requirement engineering: decomposes the requirements according to their criticality and complexity. For example:
  - Safety critical
  - Mission critical
  - Performance requirement
  - (Optional) features and customization

Asimov Laws of Robotics

- In general, a robot:
  - 1. may not injure a human being or, through inaction, allow a human being to come to harm.
  - 2. must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
  - 3. must protect its own existence as long as such protection does not conflict with the First or Second Law.

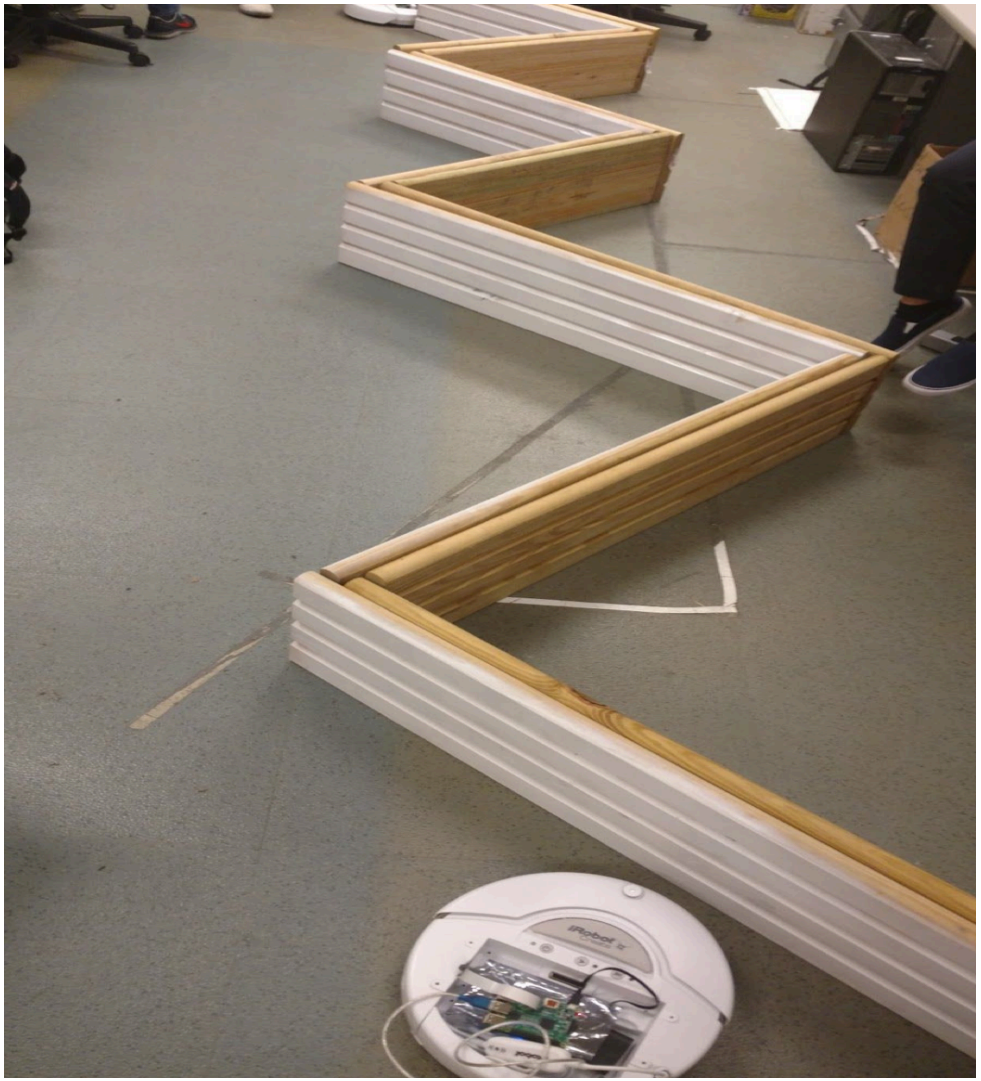
Our Laws of Robotics

- A robot may not injure a human being
- A robot must protect its own existence, except when it conflicts with First Law
- A robot must obey the orders given to it by human beings, except where such orders would conflict with the First or Second Law.
  - E.g. robot is on a scientific mission and carries some payload (camera, trajectory, external tasks, etc.)

MP2: Ancient Mine Exploration

- A farmer was planting a tree in their backyard when the ground caved in, uncovering an entry way to an ancient underground maze.
- A research team is assembled. They send a robot into the maze.
- The robot must map the maze while taking pictures of all encountered objects.
- Since this maze is labeled an archeological site, the robot must not run into or damage any object.
- The maze may include cliffs and traps. The robot must protect itself while in the maze.
- Scientists are excited and have software (external task) they want to download to the robot and run. This payload will execute analytics on data received. It may be buggy, but it needs to be executed in the maze because limited bandwidth prevents the robot from reporting all data outside.
- The robot should complete the mission as quickly as possible.

The maze



Requirement Engineering?

- **Safety**-critical requirements
  - What are the hazards from environment?
- **Mission**-critical requirements
  - What needs to be done for this mission to be successful?
- **Performance** requirements
  - What decides how well the mission is executed?

Requirement Engineering

- **Safety**-critical: basic survival/safety
  - Stop when hitting the wall, wheels dropping, cliff signal, overcurrent
- **Mission**-critical: the purpose of the mission. Violating them may cause the mission to fail, but the system itself will remain unharmed
  - Track trajectory, take pictures, run external task
- **Performance**-critical: the quality attributes that describe how well the mission is performed
  - Travel time, time to finish entire mission (including processing pictures), photo analysis

Design multi threads

- How many threads do we need?
  - Wall sensor
  - Cliff/wheel drop sensor
  - Overcurrent sensor
  - Motion control
  - Sound/LED
  - Tracking trajectory
  - Taking photos
  - Matching pictures

- **Safety**-critical: basic survival/safety:
  - Highest priority
  - Least period, fastest response
- **Mission**-critical: the purpose of the mission:
  - 2^{nd} highest priority
  - Motion control, tracking, navigation, etc.
- **Performance**-critical: the quality attributes that describe how well the mission is performed:
  - Lowest priority
  - Computer vision

- At least 3 threads:
  - Hi-pri: checking safety-related sensors, stop immediately if needed.
  - Mid-pri: following the wall, tracking trajectory.
  - Low-pri: taking pictures, run openCV.

High

```
pthread_attr_t attrSafety;
sched_param paramSafety;
pthread_attr_init (&attrSafety);
pthread_attr_getschedparam (&attrSafety, &paramSafety);
paramSafety.sched_priority = 4;
pthread_attr_setschedparam (&attrSafety, &paramSafety);
```

Mid

```
pthread_attr_t attrMotion;
sched_param paramMotion;
pthread_attr_init (&attrMotion);
pthread_attr_getschedparam (&attrMotion, &paramMotion);
paramMotion.sched_priority = 3;
pthread_attr_setschedparam (&attrMotion, &paramMotion);
```

Low

```
pthread_attr_t attrVision;
sched_param paramVision;
pthread_attr_init (&attrVision);
pthread_attr_getschedparam (&attrVision, &paramVision);
paramVision.sched_priority = 2;
pthread_attr_setschedparam (&attrVision, &paramVision);
```

Multiple threads

- How to enforce priority?
  - Set scheduling priority explicitly in code
  - Let threads sleep for different time
    - High priority sleeps less (e.g. 100ms)
    - Low priority sleeps more(e.g. 2000ms)
- Note we have external task running at lowest priority, so sleep to give CPU to external task.

Issues

- Serial port contention
  - Use mutex.
  - Remember to release mutex before sleeps.

- Split into more threads:
  - Separate cliff sensor, overcurrent sensor into separate threads?
  - Separate taking picture thread from matching picture thread?
  - Don't use too many threads, overhead.

- Camera & OpenCV:
  - MP2 allows you to take pictures in the maze and analyze them outside of maze: to save travel time.
  - You can run image matching thread later.
  - MP3 requires to take pictures and match them in the maze, you should run matching thread in the maze.
  - You can extract features of images beforehand if you want to run image matching during in the maze.

- How often to take a photo?
  - Too frequent: memory can't store them, writing to disk costs time.
  - Not frequent enough: may miss pictures.

- Where will TA put pictures?
  - Some distance away, can be seen by robots when going straight.

- Overcurrent sensor issue:
  - Overcurrent sensor may have false positives, unstable, may need to average multiple readings.
  - TA may drag robot to cause overcurrent.

- Wall sensor issue:
  - Wall sensor has readings only when almost parallel to wall, maximum value and angle differ at different distance.

Wall following

- How to prevent losing wall signal?
- What to do when signal reading is lost?
- What is the best way to start after bumping into the wall for the first time? Since entry angle can be arbitrary.

Note: timer starts after bumping the first wall.

Questions?

Start early!