



Robot Control

- Introduction

- There is a nice review of the issues in robot control in the 6270 Manual
- Robots get stuck against obstacles, walls and other robots.

Why? Is it mechanical or electronic or sensor failure?

Answer: Sometimes! More often than not it is because of POOR SOFTWARE design. The robot is not mechanically stuck, *it is mentally stuck!* The program did not account for this situation and did not provide a “way out” of the dilemma.

1



Robot Control

- Example

- Bump sensors (switches) do not always trigger!
- A robot wall follows for a while and then for no apparent reason bumps the wall.
- A robot approaches a corner and “trembles” (that is, gets trapped!)

- How long does it take to write robot software?

One or two days like in other courses?

Two factors to consider:

You are running in *real-time* and you are *multitasking*

2



Robot Control

You cannot predict every situation, further, there are always situations you never thought about when designing the software

- How often do we deal with time-varying data in our engineering curriculums?
- Can you write contest/demo day software at the same time you are modifying your platform?
- What happens if your code has “values” that were obtained by calibration (e.g., servos) and you now have to replace your servos?

3



Arroyo's Rules of Thumb

- Successful robots are those that have their platform completed 2 weeks before demo day.
- Successful robots demo at the 90-95% level on pre-demo week.
- Successful robots have software that modularly and systematically test subsystems independently
- Successful robots have software that was developed deliberately, using sound software engineering design principles and follow established conventions and hints given in class.

4



Robot Control Strategies

- Negative Feedback - it is called negative feedback because corrections decrease the error.
 - You compute an error signal
 - You make a change that is proportional to the error
 - Consider a wall-following example
 - Sense the values
 - Determine if too close or too far
 - Turn either toward the wall or away from the wall accordingly

5



Robot Control Strategies

```
Int Wall_dist(sensor_value Int)
{If Sensor_value < Too_far_threshold Return Too_far
  If Sensor_value > Too_close_threshold Return Too_close
  Return Ok}
Void Follow_wall ( )
{While (1) {
  Int Sense = Analog(ir_sensor);
  Int State = Wall_dist(Sense);
  If (State==Too_close) Turn_away( );
  else If (State==Too_far) Turn_toward( );
  else Go_straight;
}}
```

6



Robot Control Strategies

- Negative Feedback - Point
 - How sharply we turn the motors affect performance
 - The IR sensors need calibration
 - The threshold values need to be EXPERIMENTALLY determined
 - How fast we go around the loop determines performance
 - Can we change the motors smoothly?

7



Robot Control Strategies

- Open Loop Control - Figure *a priori* how long or how well you perform an action and program the robot to perform the action w/o measuring the results (i.e., no feedback).
 - Example: Use a shaft encoder to generate pulses and figure the relationship between pulses and distance
 - Requires a high degree of accuracy & predictability
 - Requires careful tuning
 - Errors accumulate (like compound interest)
 - Has not historically worked well in IMDL robots

8



Robot Control Strategies

- Feedforward Control - Attempts to predict (via measurement) parameters that affect open loop programs (like dynamically figuring *a priori* how the pulses in shaft encoders change as a function of battery voltage level and correcting “a priori” the pulse ticks vs distance measurements.)
- Another way of thinking about this is that it is like “compensating” open loop programs via “a priori” measurements.

9



Sensor Calibration

- The threshold values for the sensor levels, as measured by the A/D system (0-255) are correlated to physical units before values are coded into your software
- As a minimum these should be #define constants
- The better IMDL robots have included self-calibration or dynamically adjusted calibration routines to automatically adjust the sensors.
 - Example: Jose Diaz’s robot when first turned on, would approach a wall and dynamically adjust the Too_close / Too_far sensor thresholds.

10



Sensor Calibration

- Tae Choi's PhD work at MIL - the robot learns its own threshold values (uses machine learning).
- Light Sensors (CDS cells)
 - Affected by room lightning levels
 - Should be physically shielded
 - You must control the source of the light if the “degree of light” is important
 - You should measure the “ambient light level” is possible

11



Sensor Calibration

- Motor/Servo Sensing
 - Depend heavily on battery level, which can be sensed using a voltage divider circuit
 - You can sense battery charge by using a thermistor
 - You should measure “free”, “geared” and “stall” currents
- You should use “persistent global variables” for all your calibration values

12



Failures

- Mechanical Failures: careful design exploiting modularity and with counter-measures
- Electrical Failures: loose connections, cold solder joints, shorted leads, etc. Use good techniques and plenty of hot glue and preventive measures
- Unreliable Sensors: sensors provide “noisy” samples (the value changes given the same physical conditions) or fails to register (e.g., a bump sensor that does not trigger).

13



Sensor Failures

- Spurious Sensor Data - (can software filtering be used?) Fix via averaging or taking differences
- Missed data - electrically or because of software design you may miss reading a sample (the change may have been too fast/slow for the software to detect it)
- Corruption - battery level or a change of environment change the sensor readings
- Since filtering is equivalent to averaging it is possible to filter and detect anomalies and fix

14



Problems in Task-Oriented Control

- Robot runs into a wall, object or other robot
Respond to bump/switch sensor
- Robot runs into a wall, object or other robot
The bump/switch sensor did not trigger
- Robot wanders and never “sees” anything
How long do you wander and not see (spin in place)
- Robot slams into a wall, object or other robot
Respond forcefully to a bump/switch sensor (oscillate)

15



Problems in Task-Oriented Control

- Sensor, bump mechanism fall off or disconnects
Detect anomalous readings and stop(?)
- The board fails
- The board resets
- Electrical noise
- Multiple battery packs

16



To Handle Possible Problems in Task-Oriented Control

- Exit Conditions
 - If a robot does not sooner or later run into an obstacle, then something is wrong
 - Instead of while(1) use while (!stuck()) and return either Normal_exit or Error_exit values
- Timeout -- add a timer-based exit condition or count the number of actions. If too many, then do something random

17



To Handle Possible Problems in Task-Oriented Control

- Monitor Transitions
 - Correct sequences are determined a priori, and if you get a wrong sequence something is wrong or do something random (e.g., you cannot be following a wall if all you get is Turn_left commands...)
- To multi-task or not to multi-task
 - Task sequence (use a loop to execute tasks)
 - Allow concurrent tasks that do not compete
 - Use priority or other tricks to resolve task conflicts

18