CSC C85 – Fundamentals of Robotics and Automated Systems
Project #1 – Fault Tolerant Lander Control

The goal of this project is to give you practical experience in the writing software that is fault tolerant. You will be faced with a system whose components (like any real system) have a small probability of failure. This system is required to perform a critical task, and it must work regardless of component failures. You will write the code to accomplish this!

Learning Objectives – after completing this project you should be able to:

Design software by thinking in terms of ***what could possibly go wrong***, and building into your design the ability to resist, compensate, or otherwise handle failures.

Implement methods for detecting failures given sensor inputs and input history, as well as ***predicted*** behaviour for the system.

Implement simple methods for re-configuring your software to compensate for failed or unreliable components.

Use multiple source of information to accurately estimate state variables whose sensor readings have become unreliable or invalid.

Practice implementing code to perform sensor denoising / signal cleanup in order to recover useful information from noisy or corrupted information.

Skills Developed:

Writing code to keep track of sensor readings and detect failures

Writing code that re-configures itself to compensate for failures while remaining able to carry out its task.

Working with a system whose behaviour must be controlled in real-time regardless of failures

Reference material:

Your lecture notes on control systems, sensors, and reliable software design

The comments and explanations inside ***Lander.cpp***

Your ingenuity! Be smart and creative. Bonus points for clever solutions to tough problems.

Programming Exercise.

This is designed to run on Linux - normally we would require you to run this at the lab to show your work, and to make sure it compiles and runs on those machines. however, due to the capacity and availability restrictions on labs, it's ok if you can demonstrate your work on the laptop you're using during progress checks and for consulting/help sessions. The final submission ***has to compile and run*** on the lab machines (so we can do our testing after the deadline).

Download and unpack the starter code from the course website into an empty directory.

Unpack the starter code and build the executable.

The Problem:

The code simulates a ***Lunar Lander*** and its ***control software***. The craft is expected to land automatically on a pre-specified ***safe*** platform (coloured in red). To land safely, the craft must be aligned within ***15 degrees*** of the vertical direction, and it must be moving slower than ***10 m/s***. Breaking any of these rules, or touching any part of the terrain other than the platform will cause the craft to crash and be destroyed.

This used to be a video game. Now we will write software to land the craft automatically regardless of hardware failures!

Setup:

The code I am providing ***already contains flight control software to land the craft safely*** in both the easy and hard maps. Run the program under the no-failure mode and see how it works.

You will need to read and understand the two main routines that comprise the flight control software.

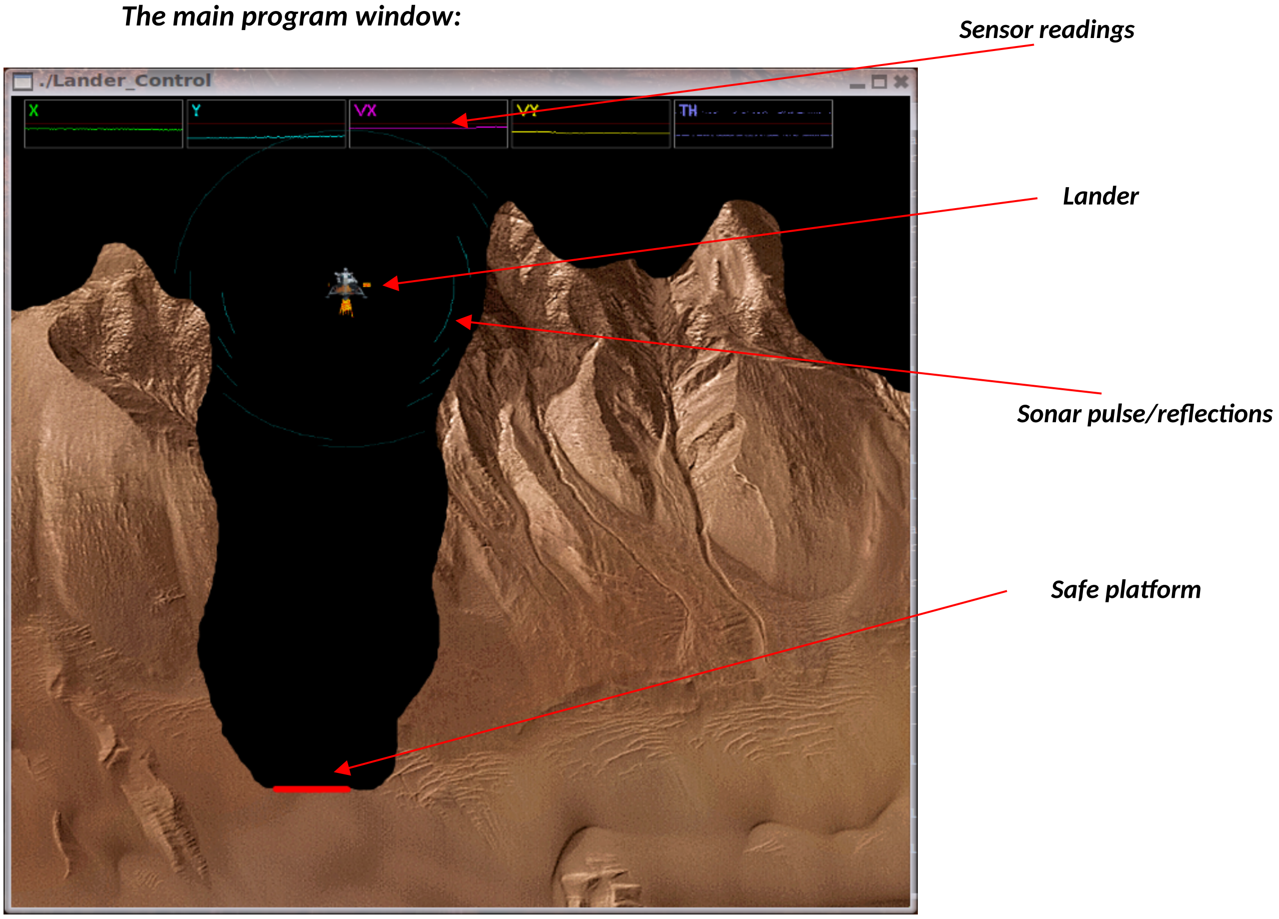
The problem with the existing software, is that it ***assumes that everything will work just fine***. When components fail, the flight control software will fail and the ship will crash,

Your task:

Re-write or improve the existing flight control software so that it ***safely lands the craft regardless of failures in ship sub-systems.***

This will involve clever use of sensors and ship controls, as well as technical prowess in dealing with noisy measurements and a real-time control system.

The main program window:



The craft has a set of sensors that can be used at any point during the simulation to determine the state of the lander:

- *Position sensors (X and Y)*
- *Velocity sensors (X and Y)*
- *Angle (theta, in degrees, w.r.t. vertical)*
- *Sonar distance from closest surfaces (360 degrees around the lander)*
- *Laser range-finder*

The sensor readings are ***noisy*** which means you will have to implement some strategy to handle noise. Noisy sensor data is displayed at the top. For X and Y the reference is the center of the map, for V_x and V_y the reference is 0, and for theta the reference is the *+x* axis at 0 degrees.

Additionally, the craft has ***4 control functions***:

- *Main Thruster control*
- *Left and Right thruster control*
- *Rotation control*

Solving the problem:

The starter code contains detailed comments on what functions to call to access the sensors and controllers. It tells you what variables and parameters you have access to, and what variables you are not allowed to change.

You are not allowed to change any part of the existing code other than the two routines that implement the craft control.

In addition to that, you ***MUST either add*** your own modified sensor functions, or add code to the craft control functions to deal with noise and sensor failures. But note that ***these functions have to use the existing noisy sensor interfaces***, i.e. you are ***not allowed to access simulation states directly. That is cheating, you get zero.***

The key here is to think about implicit redundancy in the information provided by the different sensors.

You should first try to handle failures in the control the thrusters, (note that the rotation control always works!). This should not be extremely difficult, but will require you to work with backups.

Note that the lander provides ***boolean flags*** that indicate whether each of the thrusters is operational or not.

Once you have managed to make your craft robust to failures in the control components, work on robustness to sensor failures.

These are more difficult since there are no flags to indicate when they are not working properly. You will need to write code to determine when the sensors are faulty, then find a way to compensate for their absence.

Some sensor failures are easier to deal with than others!

Document all your design decisions, strategy for solving each possible problem, and how well it works!

The code provides a variety of test scenarios. We will test your code on failure modes ***1, and 2***. But note that ***3*** is a super-mode that allows us to easily simulate any setting of 1 and 2. Test using mode 3 to make your life easier.

Compiling your code:

Use the included ***Makefile.*** This works on mathlab as well as on the computers at the Lab as well as any properly configured Linux computer. If you are having trouble, check the output for missing libraries, and feel free to come get help during offic hours!

General advice:

Do not ignore noise!

Do not ignore the past! In order to figure out where you are, you need to know where you've been.

Pay attention to the range of sensors available and use them wisely

Understand how the existing flight control software works before you start changing it. Try to think of all possible ways in which it can fail – then account for each of those.

Bonus marks will go to clever solutions. You can use any of the techniques we have discussed in class, and you're always free to look into how such problems are addressed in industry.

Remember that all team members ***must understand thoroughly*** the submitted solution and the process that led to it.

Submitting your solution (deadline is Thu. Sept. 26, 3pm):

Rename your solution to be: ***Lander_Control_teamname.cpp***

Submit ***both*** the ***.cpp*** file and the completed ***REPORT.TXT as a single .tgz*** archive, and make sure it decompresses properly!

Electronic submission on Quercus is due by the project deadline

A final word of advice:

Start early. It will probably take more time for you to read and understand the provided code than to actually solve the simplest failure modes (harder failure modes are a different matter).

Ask questions! Use the office hours, and come for mentoring!

Nothing is completely fail safe, ever. But, it can get pretty good! Try to impress us!

Project Timeline

First Week – Sep. 9 – 13

a) Compile and run the starter code – make sure you can work!
b) Read through the starter code, try the different modes of failure provided by the code, and see what happens to the lander when no fail-safe measures are implemented.
c) ***Plan and design*** your fail-safe measures for each failure mode.

Second Week – Sep. 16-20

a) Implement the functions that allow your lander to survive if thrusters fail
- Many ways to do this, but if you think it through there is one proper design that will handle all engine failures without trouble.

b) Start looking into how you will compensate or replace faulty sensors.
- Characterize the types of failures (does the sensor get more noisy? Does it simply return junk? does it stop returning anything?).
- Think whether and how remaining sensors can be used to compensate for each of the possible sensor failures.
- Start applying what we will cover in lecture regarding sensors and denoising

Mentoring: Sep 22 during tutorial
You can bring questions or consult your TA/instructor regarding your plans to handle sensor failures.

c) After the mentoring day, start working on your code to compensate for sensor malfunctions. Work on that for the rest of the week.

Third Week – Sep. 23-26 (Notice this is due Thursday, in time for the review)

a) Implement the functions that will allow your lander to survive any combination of 2 failures.
b) Test! Test! Test! - maye your lander crunchy, see if you can make it survive multiple thruster/sensor failures. Can it land with all sensors broken?

Deadline for submitting your completed code:
Thursday, Sep. 26 @ 3pm – submit only one file per team on Quercus.

Project Review: Thursday, Sep. 26 – During the tutorial/lab times
You will explain and show your work. Time-slots for each team will be arranged by Doodle poll in advance of this date. This is 80% of the mark for the project.

Code review and answers in the report file are worth 20%