MECH 223 MatLab Simulation Implementation

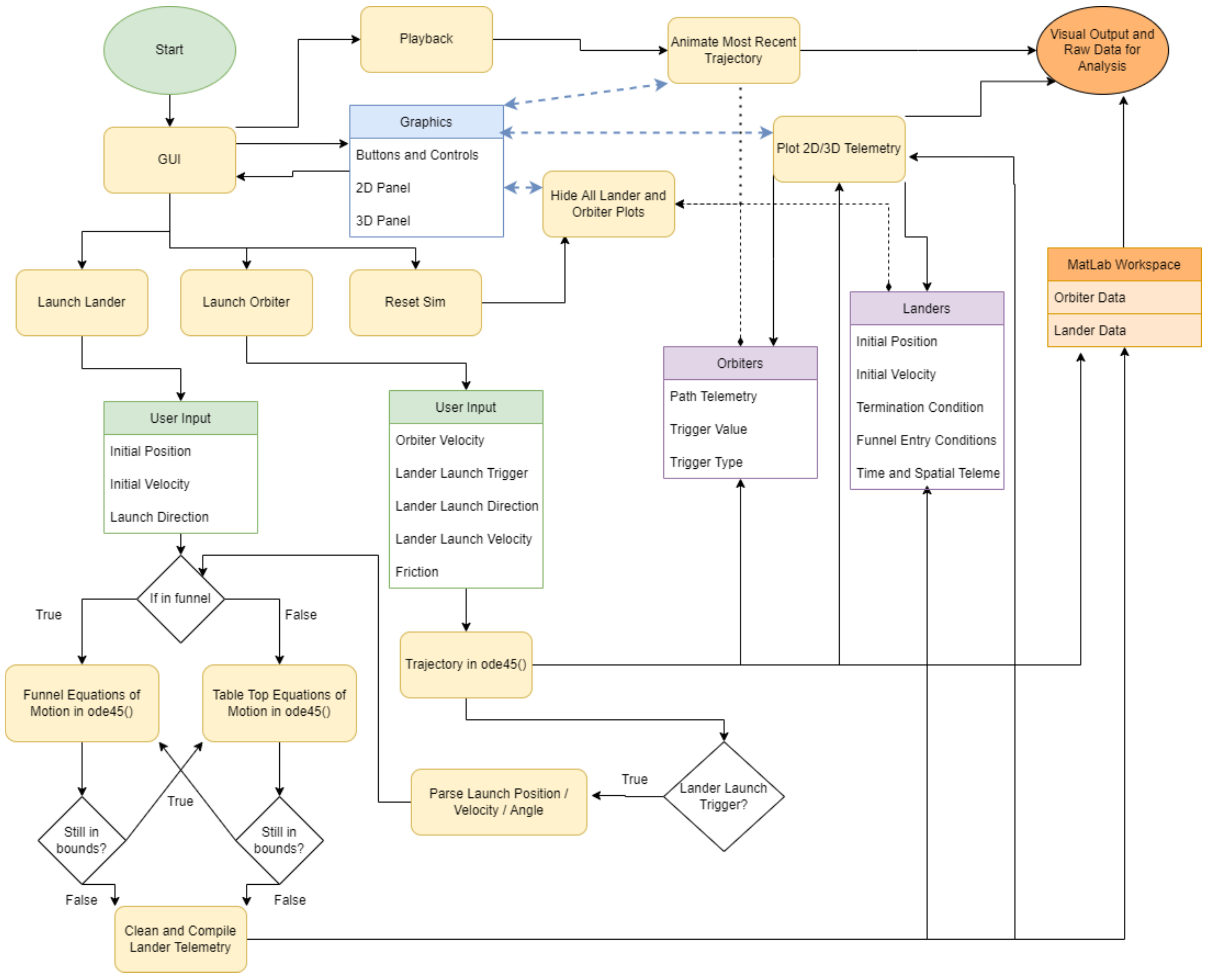

Figure 1: Simulation Top-Level Diagram

The source code and Readme file is provided on GitHub. In addition, a tutorial, demonstration, and sample outputs are available by video here.

A brief and informal overview of some of the software implementation strategies and challenges can be found below.

The program makes extensive use of structures which can be thought of as packages of variables of mixed types that can be passed and called through a field name. For example 'container.blue' will call the structure called 'container' and the field value 'blue' which can be of any type.

The GUI is implemented through some of the powerful features available in MatLab. At runtime, all of the handles for the graphical and UI elements are built and packaged into a GUI structure and then passed to the main() function. Some of the GUI elements are set to be not visible and can be activated as needed (the laser sight lines for example).

The main() function serves the purpose of allowing all of the functions contained within to share variables, access, and modify all of the variables contained in main(). In short, the scope of variables is not local for nested functions in MatLab.

The main() function stores a list of 'landers' and 'orbiters' which are arrays of the structure types that contain all of the trajectory and graphical information for launches. When an event occurs to initiate some type of launch, the trajectory is pre-calculated and trimmed according to the model and then displayed. The information is stored internally in the program and also exported to the main workspace. A better design approach would use some of the object-oriented programming features in MatLab, particularly for the lander and orbiter 'pseudo-objects' formed by using structures. Time constraints did not allow for this style of implementation, and the code will not be updated or maintained into the future.

For the lander launches, there were a few programming design challenges that came up. One of them was that the mathematical modeling system for the funnel and for the table-top are very different and could not be combined into a single implementation of ode45(). When a new lander is instantiated, it receives a tag as being active. The lander remains active until it either stops, exits the playing field, or falls out the bottom of the funnel. The tabletop and funnel trajectory calculations are implemented as two mutually-recursive functions that terminate when the ball is no longer active. The tabletop trajectory is calculated in Cartesian coordinates with an origin at the bottom left, while the funnel operates in cylindrical coordinates centered on Titan. The program can convert and pass information between the two systems to produce a final Cartesian-only trajectory analysis.

The orbiter telemetry only implements a table-top trajectory system because it does not belong in the funnel (would potentially be entertaining to see in a real competition). The framework for working with forces in both the x and y-direction is coded into the software, but our final prototype design was primarily rectilinear x-axis motion for the orbiter by design. If a lander launch event is triggered, the orbiter position and velocity, as well as the lander launch vector, are used to trigger a new lander launch in the correct location and velocity. The orbiter and lander are not explicitly paired in the current implementation.

The playback animation system uses an algorithm to slice the trajectories into evenly spaced frames. There is a use of the tic-toc system to throttle the speed of rendering so that it plays events back in real-time, as close as possible. The coordinates of the frame slices are continuously updated to the graphical objects stored in the GUI structure representing them. The graphics are redrawn each frame to achieve a fluid animation effect.