ENPH 213 Project 2020

Final Report – Projectile Game Theatre

2020-04-15

Csaba Nemeth

Queen's University, Kingston ON

Student ID: 20090753

Section 1 Background – Projectile Modelling

The modelling of a projectile in a flight is most commonly described using the analytical equations of kinematic motion, namely that, for a given initial velocity v_0 and initial position s_0, the position of a particle undergoing constant acceleration a is given as a function of time t.

$$\vec{s} = \vec{s_0} + \vec{v_0}t + \frac{1}{2}\vec{a}t^2 \tag{1}$$

Equation (1) describes the shape of a parabola, and while accurate within a vacuum, it diverges from the true projectile path in the presence of aerodynamic drag and buoyancy. This divergence typically occurs at a location between 10% and 70% of the total path length, but can vary greatly depending on the characteristics of the projectile in question [1]. Examples of such trajectories are shown in Figure 1.

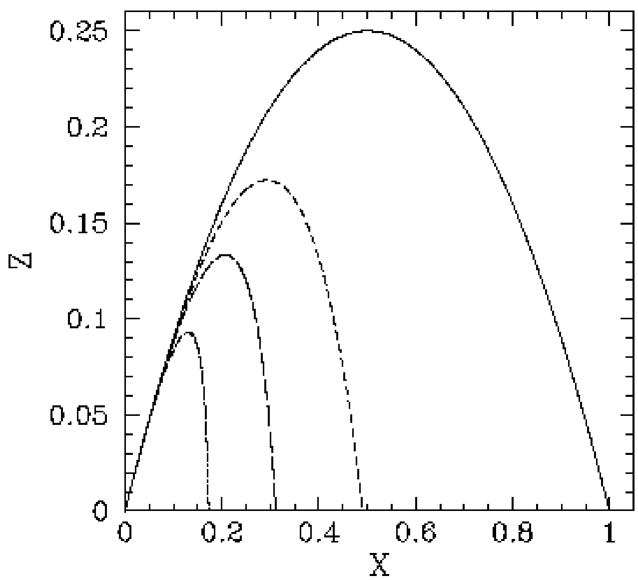

Figure 1: Various projectile trajectories, indicated with a dashed line, plotted against a trajectory calculated using Equation (1). Height Z is plotted against horizontal distance X, relative to the total path length. Citation: [2].

This figure indicates that another approach is needed to describe the path of a projectile for the game theatre simulation.

Section 1.2 Modelling with Drag and Buoyancy

The force of drag F_D of an object traveling through a fluid is given by:

$$F_D = \frac{1}{2}\rho|v|^2 C_D A \tag{2}$$

where ρ is the density of the fluid, $|v|$ is the magnitude of the projectile velocity relative to the fluid, C_D is the coefficient of drag, and A is the cross-sectional area of the projectile [3]. The acceleration due to drag in a cartesian component x, y, z, is given by:

$$a_{D_{x,y,z}} = \frac{F_D}{m}\frac{v_{x,y,z}}{|v|} \tag{3}$$

where m is the mass of the projectile. Finally, the force of buoyancy acting on a particle is given in Equation (4):

$$F_B = \rho g V_f \ [+\hat{z}] \tag{4}$$

where g is the acceleration due to gravity, and V_f is the volume of the displaced fluid. In this case the buoyancy force acts in the positive z direction, opposing the force of gravity. The above forces must be considered when modelling the flight path to avoid the divergence of the solution. This model can be constructed analytically, as Seungyeop Han outlines in a journal article published to the International Federation of Automatic Control journal [4]. These equations of motions are accurate to within 0.01% of the true projectile position but require precise knowledge of environmental conditions and projectile parameters [4]. This is not a viable option for the simulation because the model must be able to accurately predict positions of projectiles purely from positional, velocity, and acceleration data.

Figure 1 and Equation (1) indicate that while divergence occurs for relatively large time-intervals, over a small finite interval dt, Equation (1) is accurate and reduces to a linear form. This means that the projectile path can be found computationally by solving for the buoyant and drag force at each timestep and using the resultant velocity and acceleration vectors in Equation (1) over an interval dt. As shown in the following section, this also allows the effects of drag and buoyancy to be deduced from purely positional data and was therefore the selected method.

Section 1.2.1 Deriving Acceleration and Velocity from Position Data

Observations from the game server regarding positional data are taken once every second. To be able to accurately predict the path of a projectile, a descriptor is needed of the unknown projectile parameters that encompass the mass, density, cross sectional area and drag coefficient. Let us call this descriptor A_{proj}. To derive A_{proj} from the positional data, the velocity and acceleration of the projectile is needed at a given point. The velocity at the central observation can be found simply by subtracting the relative position vectors and dividing by the time between observations. An average of the two vectors gives a reasonable estimation. The acceleration can be found in the same way with the two calculated velocity vectors. While this method is crude, it will be shown later that it is accurate enough for the purposes of the simulation.

Without considering the effects of drag and buoyancy, the expected vectors of a ballistic projectile are:

$$\overrightarrow{v_{exp}} = \langle v_x, v_y, v_z \rangle$$

$$\overrightarrow{a_{exp}} = \langle 0, 0, -9.81 \rangle$$

Whereas the actual acceleration vector a_0 will be:

$$\overrightarrow{a_0} = \langle \ \frac{F_D}{m}\frac{v_x}{|v_0|}, \frac{F_D}{m}\frac{v_y}{|v_0|}, -9.81 + \ F_B + \frac{F_D}{m}\frac{v_z}{|v_0|} \ \rangle$$

$$F_D = \frac{1}{2}\rho|v|^2 C_D A$$

$$F_B = \rho g V_f$$

which considers the drag and buoyancy effects.

We can solve these equations for A_{proj} independently through the x and y components of acceleration, and taking their average gives a best estimate:

$$A_{proj} = -\frac{a_{0x,y}}{v_{0_{x,y}}|v|} = \frac{1}{2}\frac{\rho C_D A}{m} \tag{5}$$

This means that equation (3), the acceleration due to drag is then simply:

$$a_{Dx,y,z} = -A_{proj} * v_{x,y,z} * |v| \tag{6}$$

This can be used to solve for the acceleration for a certain time within our time-step model. This also leads to the observation that the buoyancy can simply be represented as an effective gravitational acceleration, assuming that the density of air does not change throughout the simulation. The effective gravitational acceleration is therefore the addition of the buoyant force and g:

$$g_{eff} = -a_{0z} - A_{proj} * |v_0| * v_{0_z}$$

$$g_{eff} = -9.81 + \rho g V_f$$

These constants can now be used to model the trajectory of target projectiles using three position observations at n, and the description of motion Equation (1), for each timestep n+1 is:

$$\overrightarrow{s_{n+1}} = \overrightarrow{s_n} + \overrightarrow{v_{n+1}}(dt) + \frac{1}{2}\overrightarrow{a_{n+1}}(dt)^2 \tag{7}$$

$$\overrightarrow{v_{n+1}} = v_n + \overrightarrow{a_{n+1}}(dt) \tag{8}$$

$$\overrightarrow{a_{n+1}} = \langle -A_{proj} * (\vec{v}_{x_n,y_n,z_n} - \overrightarrow{v_{wind}}) * |\vec{v} - \vec{v}_{wind}| \rangle - \langle 0, 0, g_{eff} \rangle \tag{9}$$

Where a correcting term has been added for the wind velocity v_{wind}.

Section 2 Method – Determining Launch Solution

Now that the path of the target projectile can be modelled, it is necessary to determine the conditions of the interception projectile needed to successfully collide with the target. Of these, the largest impact on the projectile motion is the initial launch velocity [2]. This is done through a bisection search, a root-finding method which is discussed in detail in Section 2.2 Launch Velocity. The following focuses on the other projectile parameters: mass, density, cross-sectional area, and drag-coefficient.

Section 2.1 Projectile Parameters

The model developed in Section 1.2.1 is most accurate for projectiles experiencing low drag and low buoyancy. This is due to the finite time step necessary for the solve. As can be seen in Equation (7), velocity is propagated linearly with the timestep dt, while the acceleration vector (containing the components of buoyancy and drag) is propagated with the square of dt. This has the implication that a high-drag model will diverge quicker from the true path with a fixed dt, leading to a more accurate prediction for low drag projectiles. This concept is explored in depth in Section 3 Analysis – Determining Launch Solution, but for the time being, it is enough to know that for an accurate interception, the non-kinematic effects of the projectile should be reduced.

This minimization can be achieved for both buoyancy and drag as they are monotonic functions for the projectile conditions, as can be seen in Equation (3) and Equation (4). Normally, it would be optimal to consider the effects of velocity on the drag, but because the bisection search breaks down in higher dimension searches, as only one variable can be varied [5]. This was chosen to be the velocity, as variations in its value correspond to the largest freedom for the bisection search algorithm [1]. It is evident that a minimal coefficient of drag corresponds to the smallest drag force, therefore the smallest allowable value of 0.05 was chosen.

While not directly optimizing for velocity, the launched projectile has maximum initial kinetic energy of 5000 [J], therefore, there is a tradeoff between the speed of the initial launch and the mass of the projectile. Furthermore, an increased mass increases the cross-sectional area, which increases the drag on the projectile. Because there is a finite amount of projectile mass available, a reasonable upper cap on the limit is 5kg, which corresponds to 8 available projectiles throughout the simulation. Because we want to maximize the range of the interceptor projectile, it makes sense to plot distance as a function of mass for a 5000 [J] launch, as seen in Figure (2).

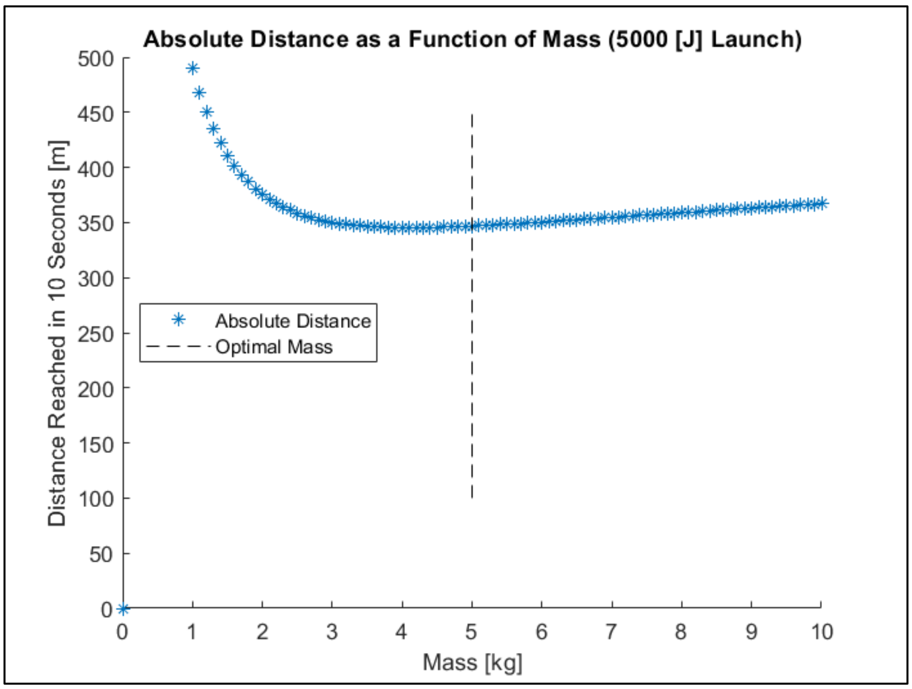

Figure 2: Absolute distance reached for a lead projectile for various masses. The optimal/chosen mass is indicated with a dashed line. The projectile path was solved computationally with a time-step of 0.01 [s].

Lead was chosen as the material of choice as it mimics the real world and is relatively dense. The shape of this curve indicates that for masses greater than 2 [kg], the absolute distance reached mostly levels out. For a spherical projectile, the cross-sectional area A is proportional to the mass m by: $A \propto m^{2/3}$, therefore, an increase in mass will decrease the drag by a larger factor than the cross-sectional area. This stipulates that we should choose the largest mass within the level area, as at that point the velocity is low enough to be reasonably efficient ($F_D \propto v^2$) and the mass is large enough to be significantly decreasing the drag. Therefore, a good choice would be the upper cap of 5 [kg], which corresponds to a cross sectional area of 0.007 [m^2] for lead. This also gives a volume of 0.0005 [m^3], which is small enough to only change the effective gravity by 0.06% away from the accepted value of 9.81 [m/s^2]. These choices minimize the non-kinematic effects of the projectile motion.

Section 2.2 Launch Velocity

To determine the launch velocity $\overrightarrow{v_0}$, a bisection search was implemented to find an intersection between a target projectile path and a projectile fired from the position of the launcher. By rewriting the intersection of two non-linear curves as a single curve intersecting the x-axis, the problem can be reduced to a root finding problem [5]. An issue that arises from using a bisection search is that an intersection point must be specified, which in the case of the simulation corresponds to some collision time t_C. Choosing this collision time depends on many factors such as the current location of the target and the velocity. The choice of this time t_C is explained in Section 3 Analysis – Determining Launch Solution

Because computations had to be completed while projectiles were detected mid-air, it was crucial that time was not wasted either when modelling a projectile or completing the bisection search. An analysis was completed on both methods to determine reasonable values for iteration.

Section 3.1 Analysis of Time-Step

The method developed in Section 1.2.1 was to write a function *modelTrajectory.m* that returned data sets of x, y, z positions at each interval dt, and a third order polynomial fit describing the path for each direction. Two methods were used to compare the relative accuracy of using a polynomial fit against a straight data set. To characterize the behavior of high and low drag projectiles, Figure 7 and Figure 8 were created, which show various projectile paths computed at values of dt. They can be found in Appendix A: Time-Step Plots. Here, it is important to notice that a low-drag projectile will converge upon a solution at a larger time-interval, while a high-drag model will require a smaller time-step and, thus more computing time to achieve an accurate result. This is the reasoning behind designing the projectile parameters in Section 2.1 Projectile Parameters to reduce the non-kinematic effects of drag-and buoyancy.

To choose a time-step, *modelTrajectory.m* was run on a dataset containing 3 azimuth and elevation data points for 84 projectiles across 3 seconds. All 84 projectiles also had a known modelled position 26 seconds after the initial data point. The returned x, y, z positions, as well as the polynomial fits for the data were evaluated at 26 seconds, and the absolute distance between the known positions and the modelled position was averaged for all 84 projectiles. This was repeated for various time-steps, and the results are plotted in Figure 3. Figure 3 shows that a time step of 0.05 is the most accurate, which was surprising considering it is quite large, and would not be computationally heavy. This meant that a value of 0.05 was chosen for the time-step.

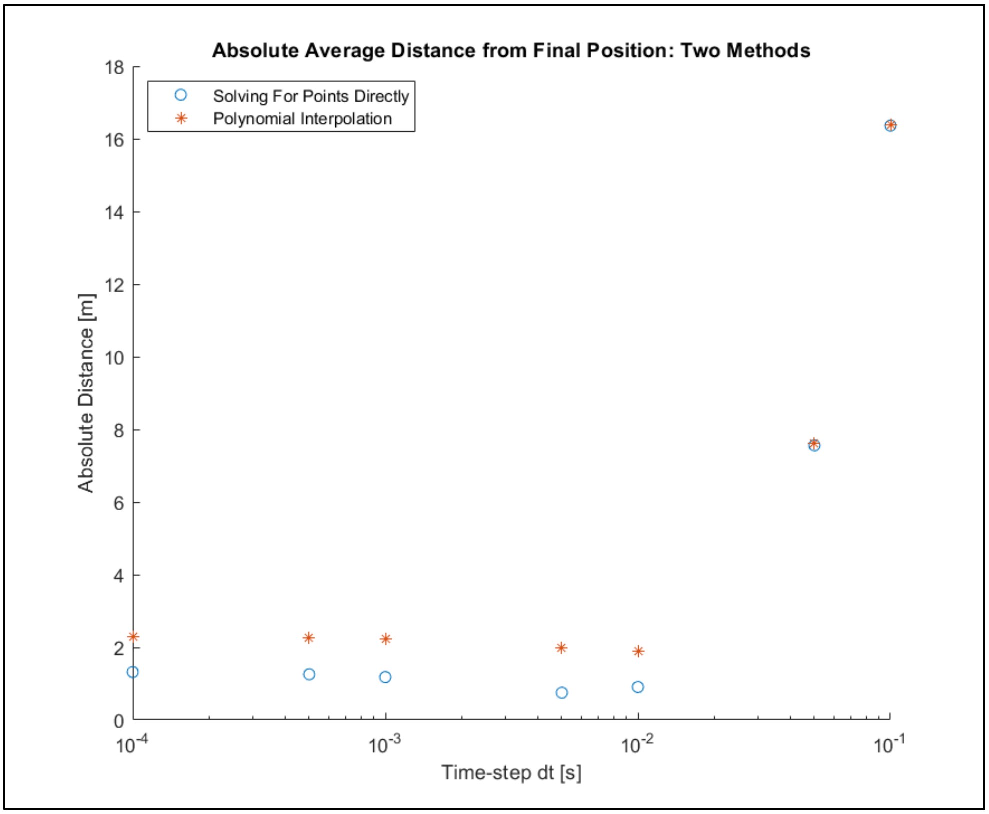

Figure 3: Average absolute distance from target projectile for various time steps, for both polynomial interpolation and direct data solve.

Section 3.2 Analysis of Bisection Search Run-Time

To analyze the effect of added accuracy on run-time, the bisection search algorithm *bisectionSearch.m* was timed and repeated for various iterations *N.* The bisection search was completed for various projectile configurations, and the values were averaged to produce the plot seen in Figure 4. The plot shows the relative accuracy of the final root velocity to the initial bisection, and the accuracy starts to level out at about 0.4 seconds of runtime, which corresponds to 12 bisections. For 12 iterations, the relative accuracy between the initial guess is 0.02%, which was too high considering it could contribute to approximately 10 [m] of uncertainty. To achieve a relative accuracy of 0.002%, or 1 [m], 16 iterations were needed which produced an average run-time of 0.6 seconds. This length was acceptable, because after the bisection search was completed the rest of the code was not computationally heavy to launch the projectile and could be completed in under 0.2 seconds. Polynomial interpolation performed worse than not interpolating the data and was this discarded.

.

The bisection search works by continuously halving an interval $[x_a, x_b]$ which contains a function $F(x)$ with a root x_0. The interval is halved, and the function is evaluated at the new location x_m. If $F(x_a)$ and $F(x_m)$ are opposite in sign, then the root x_0 must be to the left of x_m and the interval is reduced to $[x_a, x_m]$. If they are equal in sign, then the interval is reduced to $[x_m, x_b]$. This is repeated for *N* iterations until a desired accuracy is reached.

This process can be applied to a projectile search by defining $\overrightarrow{s_L}$ as the position of the launcher and $\overrightarrow{s_P}$ as the position of the target projectile at time t_C. The function $F(v)$ is defined using the model presented in Section 1.2.1 Deriving Acceleration and Velocity from Position Data, which returns the position of a theoretical projectile launched with some velocity at time t_C. The interval $[v_a, v_b]$ is defined by the velocity in the x, y, z component that corresponds to an initial kinetic energy of 5000 [J]. The midpoint of the interval v_m is evaluated in the projectile modelling function $F(v)$, and the resultant position $\overrightarrow{s_m}$ is compared with $\overrightarrow{s_P}$. The velocity interval is adjusted, and the process is repeated until $\overrightarrow{s_m}$ and $\overrightarrow{s_P}$ agree within some error, or the maximum number of iterations is reached. The error that $\overrightarrow{s_m}$ and $\overrightarrow{s_P}$ had to agree within was set as 2 [m]. This value was arrived at by looking at the average error on the modelling function $F(v)$, which was $\simeq 1$ [m]. Because this function had to be used twice, once for determining the position of the target and once for determining the position of the intercept projectile (during the bisection search), the errors were added to reach a maximum. The velocity interval was capped at ± 44.7 [m/s], which corresponds to a 5000 [J] launch.

Section 3 Analysis – Determining Launch Solution

Because computations had to be completed while projectiles were detected mid-air, it was crucial that time was not wasted either when modelling a projectile or completing the bisection search. An analysis was completed on both methods to determine reasonable values for iteration.

Section 3.1 Analysis of Time-Step

The method developed in Section 1.2.1 was to write a function *modelTrajectory.m* that returned data sets of x, y, z positions at each interval dt, and a third order polynomial fit describing the path for each direction. Two methods were used to compare the relative accuracy of using a polynomial fit against a straight data set. To characterize the behavior of high and low drag projectiles, Figure 7 and Figure 8 were created, which show various projectile paths computed at values of dt. They can be found in Appendix A: Time-Step Plots. Here, it is important to notice that a low-drag projectile will converge upon a solution at a larger time-interval, while a high-drag model will require a smaller time-step and, thus more computing time to achieve an accurate result. This is the reasoning behind designing the projectile parameters in Section 2.1 Projectile Parameters to reduce the non-kinematic effects of drag-and buoyancy.

To choose a time-step, *modelTrajectory.m* was run on a dataset containing 3 azimuth and elevation data points for 84 projectiles across 3 seconds. All 84 projectiles also had a known modelled position 26 seconds after the initial data point. The returned x, y, z positions, as well as the polynomial fits for the data were evaluated at 26 seconds, and the absolute distance between the known positions and the modelled position was averaged for all 84 projectiles. This was repeated for various time-steps, and the results are plotted in Figure 3. Figure 3 shows that a time step of 0.05 is the most accurate, which was surprising considering it is quite large, and would not be computationally heavy. This meant that a value of 0.05 was chosen for the time-step.

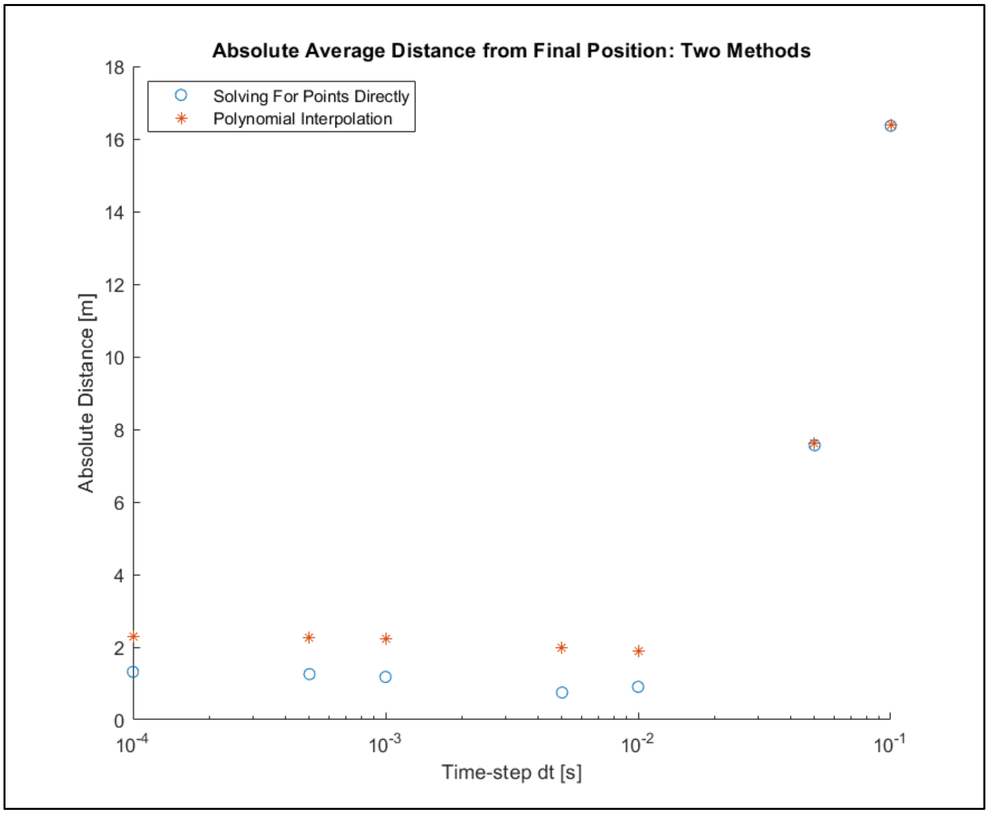

Figure 3: Average absolute distance from target projectile for various time steps, for both polynomial interpolation and direct data solve.

Section 3.2 Analysis of Bisection Search Run-Time

To analyze the effect of added accuracy on run-time, the bisection search algorithm *bisectionSearch.m* was timed and repeated for various iterations *N.* The bisection search was completed for various projectile configurations, and the values were averaged to produce the plot seen in Figure 4. The plot shows the relative accuracy of the final root velocity to the initial bisection, and the accuracy starts to level out at about 0.4 seconds of runtime, which corresponds to 12 bisections. For 12 iterations, the relative accuracy between the initial guess is 0.02%, which was too high considering it could contribute to approximately 10 [m] of uncertainty. To achieve a relative accuracy of 0.002%, or 1 [m], 16 iterations were needed which produced an average run-time of 0.6 seconds. This length was acceptable, because after the bisection search was completed the rest of the code was not computationally heavy to launch the projectile and could be completed in under 0.2 seconds. Polynomial interpolation performed worse than not interpolating the data and was this discarded.

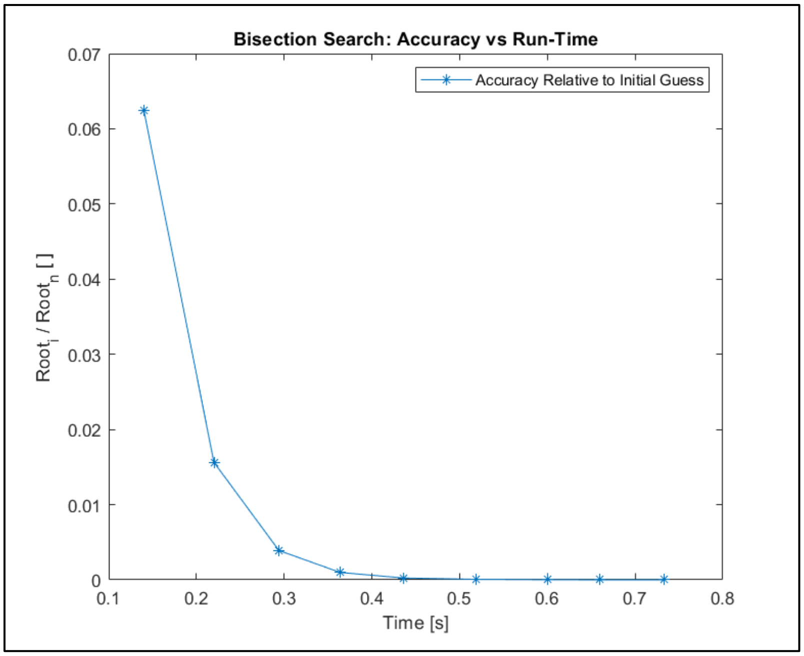

Figure 4: A plot of relative accuracy against the time taken to run bisectionSearchVelocity.m. The relative accuracy is between the initial bisection and the bisection on which the program finishes on after N iterations.

Section 4 Game Strategy and Code Design

The code was fully automated to perform everything away from human interaction and is controlled through a master function *master.m*. The decision was made to automate everything as a challenge, and to mimic real life interception programs that rely on continual observation. This function runs a continuous while loop that contains 4 stages triggered by flag variables, which call on both internal and external functions. A logical flow schematic is presented in Figure 5.

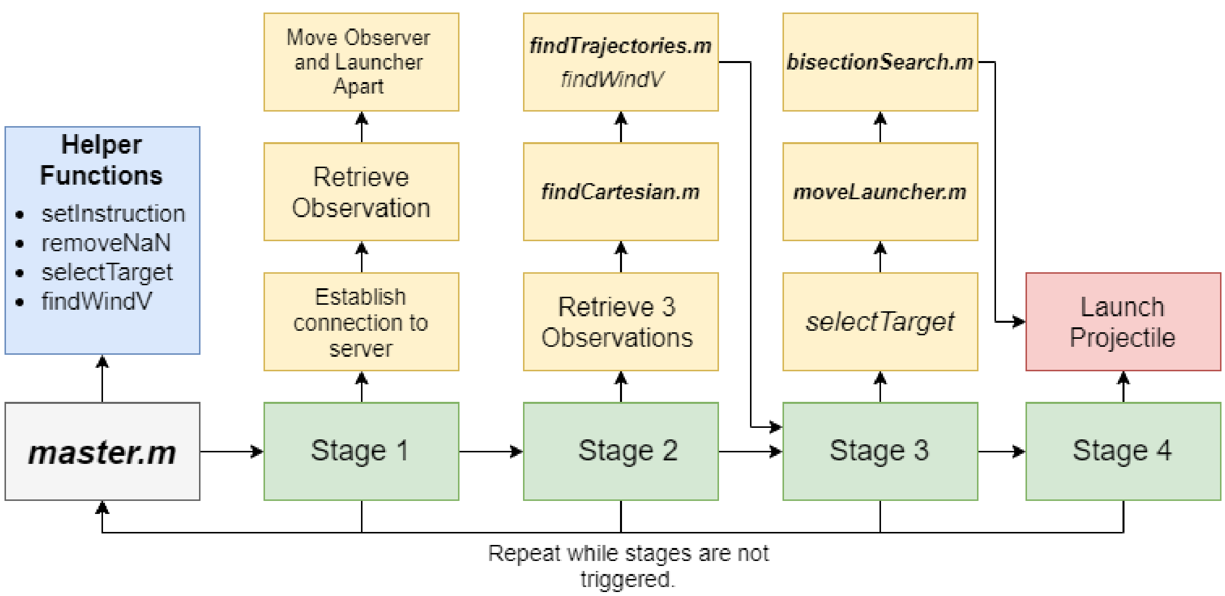

Figure 5: Software schematic of the process used to launch an interception projectile.

The first thing done in the program is to move the launcher and observer apart. This is done so that separate azimuth and elevation data from the projectiles currently in the air can be read into *findCartesian.m* and converted to (x, y, z) coordinates. After three observations spanning three seconds have been recorded, the wind velocity is computed using the observation balloons, and the trajectories of all projectiles are modelled for 10 seconds. Using these models, the target is selected, and the path is recalculated until a selected collision time t_C.

Section 4.1 Target Selection

Target selection is completed using a scoring system applied to a set of projectiles. The system was tested against a human who intuitively was choosing the best projectile to fire at and was able to agree with the human for 80% of the rounds. While this is most certainly not the best way to accomplish target selection, it factors in the three most important aspects: absolute distance from launcher, current height, and current velocity.

Consider a list of N projectiles given by the set $\{P_1, P_2, \dots , P_{N-1}, P_N\}$. The list is sorted in in ascending order by absolute distance from the launcher at 10 seconds in the model, this transformed list (A) is given by $\{P_1', P_2', \dots , P_{N-1}', P_N'\}$. At the same time, another list (B) is sorted by the current height of the projectile in the z-coordinate, given by the list $\{P_1'', P_2'', \dots , P_{N-1}'', P_N''\}$. The index position of the projectile is now counted as its rank. The initial rank is given by the index position in list A. The vertical velocity of each projectile is calculated. If the velocity if the projectile is currently upwards, then the score is given by:

$$score = rank_A - (N - rank_B) \tag{10}$$

If the velocity if the projectile is currently downwards, then the score is given by:

$$score = rank_A - rank_B \tag{11}$$

The projectile with the lowest score is chosen. Preference is given to those projectiles that are currently moving upwards and are at the earliest stage (lowest z) of their path and a scoring system ensures a projectile will always be chosen.

Section 4.2 Launcher Movement

Launcher movement is one area that needs to be further developed. Currently, after the target is selected the launcher is moved at maximum velocity towards the landing location. This can only be done for a few seconds before the bisection search takes over and the launcher must be halted. A better approach would to be start moving the launcher as soon as the first observation is taken and keep changing directions as the target is progressively narrowed down.

Section 4.3 Collision Time Selection

Without the use of an optimization function to select the best time of impact, it is difficult to properly select a time of collision t_C due to the multitude possible conditions that may be experienced. To give the interception projectile as much time as possible, a time is always chosen that is 3 seconds prior to the time when the projectile will impact the ground. Through discussion with other members working on the project, and through testing using an offline data-set, 3 seconds was usually a good choice for hitting a projectile.

Section 5 Results and Discussion

The function was unable to undergo formal completion scenarios, and due to bugs and server connection issues the code was unable to complete a full round of live testing. Due to this a numerical table of results is not presented in this report. The components of the program were tested individually, with promising results for integration.

The function *findTrajectories.m* was able to predict the path of observed projectiles consistent within 0.2-0.7 meters of the actual target. Vectorization of the code meant this could be completed quickly and efficiently, and an accurate model could be developed for all projectiles of interest.

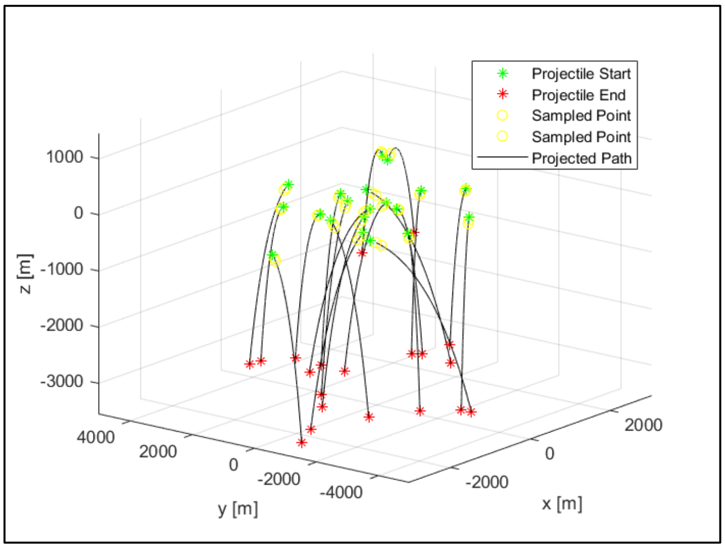

Figure 6: Modelled paths of projectiles, compared with actual position at t=26 seconds.

Connection with the server was successfully reached, and live observations were modelled, and the launcher was positioned. Target selection performed very well and was able to match a human's intuition 80% of the time. The *bisectionSearch.m* worked well and was tested using another classmates game theatre code.

References

[1] B. V. Liengme, SMath for Physics, Morgan & Claypool Publishers, 2015.

[2] R. Fitzpatrick, "Projectile Motion with Air Resistance," University of Texas, 2011. [Online]. [Accessed 4 April 2020].

[3] S. Maxemow, "That's a Drag: The Effects of Drag Forces," *Undergraduate Journal of Mathematical Modelling,* vol. 2, no. 1, 2009.

[4] M.-C. H. Seungyeop Han, "Analytic Solution of Projectile Motion with Quadratic Drag and Unity Thrust," *ScienceDirect,* vol. 49, no. 17, pp. 40-45, 2016.

[5] D. F. Richard Burden, Numerical Analysis: 2.1 The Bisection Algorithm, PWS Publishers, 1985.

Appendix A: Time-Step Plots

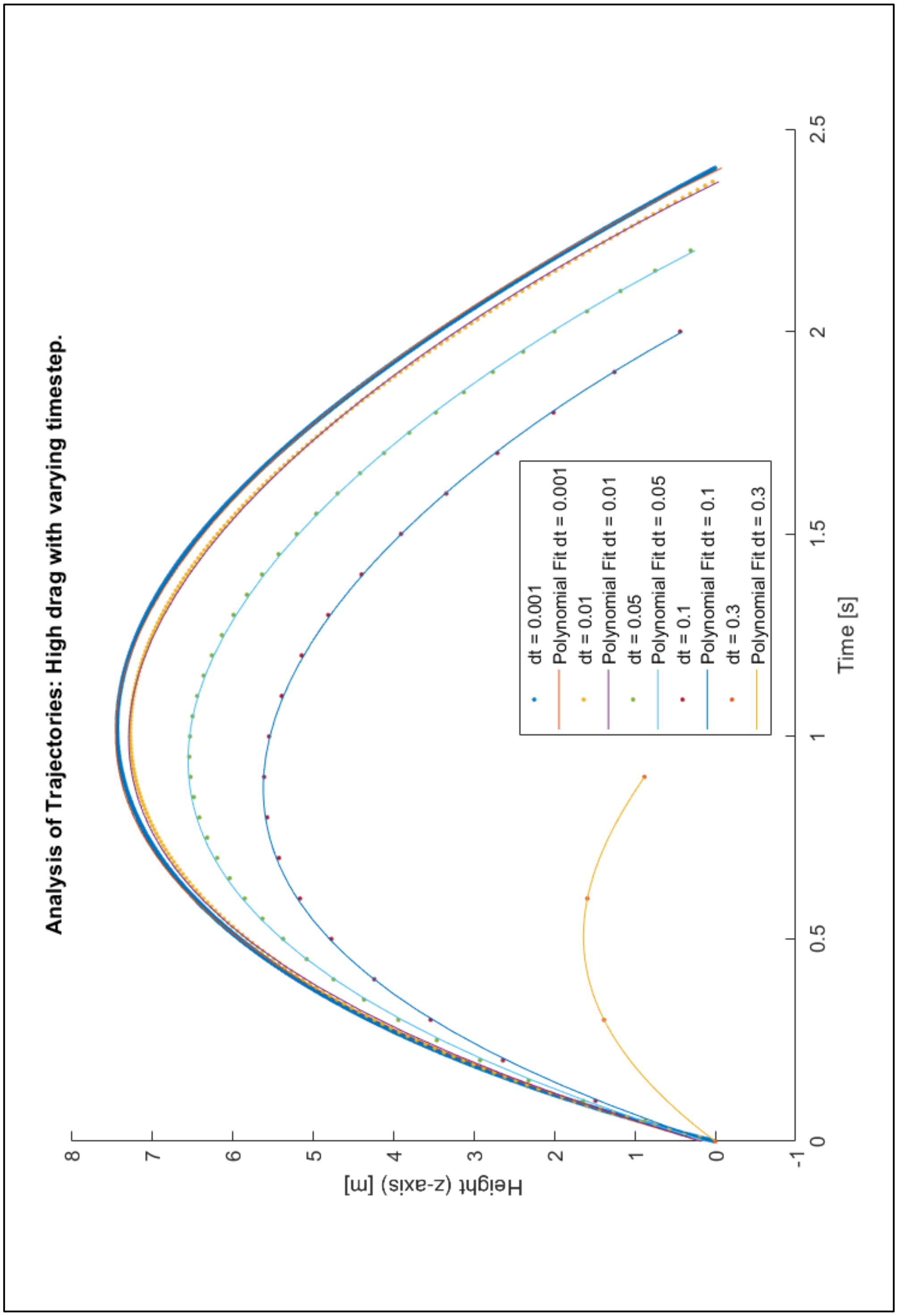

Figure 7:High drag projectile models calculated using modelTrajectory.m for various time-steps.

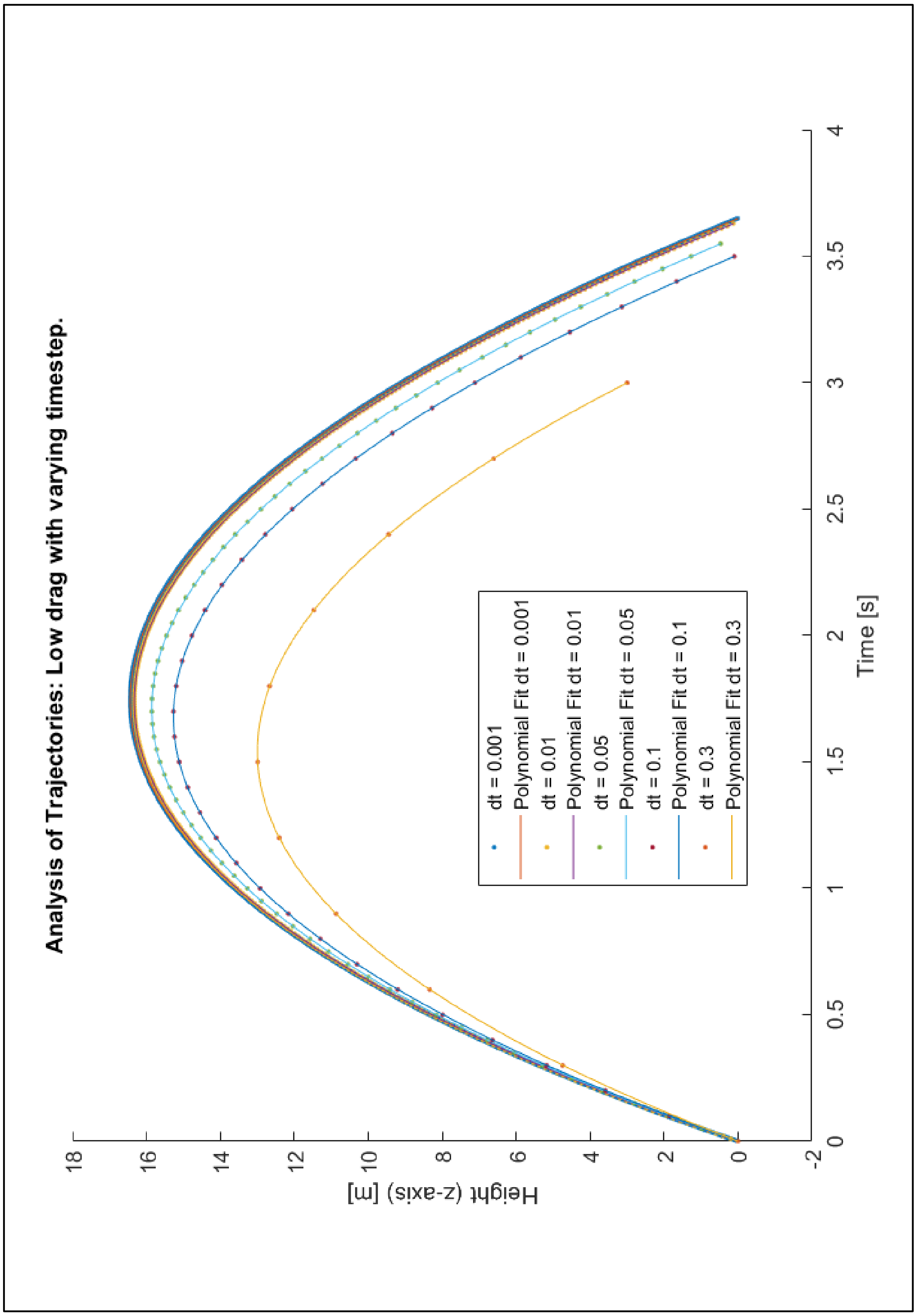

Figure 8 Low drag projectile models calculated using modelTrajectory.m for various time-steps.

Appendix B: Code