Guided Water Rocket

Gongmyung Lee, Jungbae Sohn (Daniel), Kairen Wong

I. Introduction

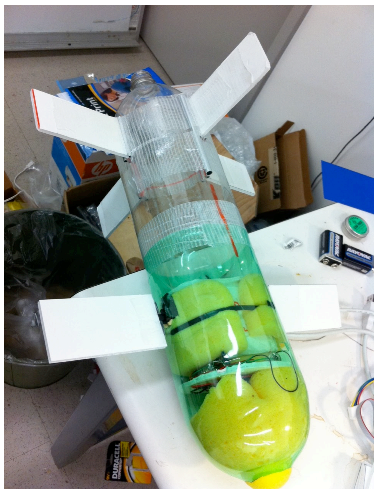
The water rocket is a well-known educational toy: a sealed soda bottle filled partway with water is then pressurized with air. When the seal is removed, the pressurized air within the rocket quickly expels the water inside, launching the rocket upwards. In our final report, we describe a simple yet successful application of reinforcement learning to water rocket flight, particularly the stabilization of the craft necessary for directed flight to take place. We also detail the design of our rocket, focusing on the electronics that allow the rocket to transmit information to a wireless receiver, as well as the machine learning algorithms applied to reduce noise during launch.

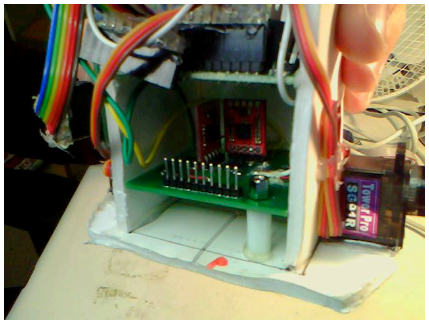
The appeal of the water rocket is twofold. First, it is "simple" to build and receives only one thrust, simplifying the engineering process. Second, the modeling process surrounding a water rocket is complicated. We know that values such as the pitch, roll, yaw and location of the rocket can help us determine where the rocket will be next, but the intimate control of fins required for such flight escapes us and would require many hours of manual practice ("biological reinforcement learning") with the controls. Though the water rocket has been quite thoroughly examined, in many cases allowing for detailed simulators to be built, the application of machine learning algorithms to controlling its flight still has greater implications; one imagines the exportability of a well-designed algorithm to autonomous fixed-wing aircraft and missiles.

II. Building the rocket

The rocket is divided in to two parts: the fuel tank and the payload. The fuel tank consists of an intact soda bottle and four orthogonally placed stabilizer wings that will keep the rocket from spinning too much and provide adequate drag so that the rocket can guide itself head first.

The payload stage contains of the brain and muscle of the rocket: the microcontroller, GPS, Xbee, inertial measurement unit, and two RC servo motors serving as the control. The inertial measurement unit is carefully collocated at the center of mass of the rocket when it is empty to avoid unnecessary non-linearity in measurements. Three gyros and one 3-axis accelerometer are fused together to give an accurate measure of the rocket's attitude. The 10Hz GPS module from sparkfun which uses UART to communicate with the microcontroller. Due to limited time constraint, a decision was made to use open source GPS parsing library from Cornell. This module will provide altitude, longitude, latitude, time, and crude measure of heading and velocity. Atmega128, the microcontroller of the rocket, receives information from the GPS, IMU(inertial measurement unit), and PC's matlab via XBEE to calculate the appropriate control laws and move the two canard fins appropriately using mini RC servos. The entire system hardware, with protection fuses included, is complete and running with a 9V battery power supply.The entire payload is housed in a Styrofoam box to absorb the shock of the impact. More sponge was added to fill the void between the soda bottle and the Styrofoam box to enhance shock survival.





Most obstacles in this stage lay in the complete fail-safe assembly and integration of different software and hardware modules, each with different interface protocols and operating voltage requirements, made the hardware implementation a project within a project. A couple of design flaws in circuitry (poor choice of communications logic converting transistor burning itself whenever the serial data rate goes up more than 9600 baud rates) were detected and debugged. Unfortunate short circuits disabled flight computer block 1 while block 2 was thoroughly burnt out by unexpected reverse current generated by RC servo motors during the impact of landing. After a series of modifications, flight computer block 3 was ready after the thanksgiving break to finally support machine learning. This version survived total 19 head on landing impacts and is still running.

The rocket communicates to PC via wireless and MATLAB receives the serial data packets containing pitch, roll, and yaw data real-time. MATLAB's role was to compute and update the MDP and sends simple two number commands to the rocket: left and right RC servo angle commands.

III. Filtering Noise in Sensor Readings

Since we expect the vibration of the rocket's launch and the drift of the gyros to seriously corrupt the attitude calculation, we use a discretized version of Kalman filter that we derived by hand. Given $X_{k+1} = AX_k + w$ and $y_k = HX_{K+1} + v$, with v and w as distinct Gaussian noise, we start out to compute Kalman gain K such that out end result of the calculation $\hat{X}_{k+1} = A\hat{X}_k + K(y_k - HA\hat{X}_k)$ is optimal in the sense that the cost function $E\left\{\left(X_{k+1} - \hat{X_{k+1}}\right)\left(X_{k+1} - \hat{X_{k+1}}\right)^T\right\}$ is minimized (covariance of estimation). Plugging in the defined values above to the cost function, we get

$$E\left\{\left((A - KHA)\,X_k + w - \tilde{K}HW - Kv\right)\left((A - KHA)\,X_k + w - \tilde{K}HW - Kv\right)^T\right\}$$

By expanding and applying expectation operators, and noting that v and w are statistically independent Gaussian noise with zero mean, the cost function simplifies to:

$$(A - KHA)\,P_k\,(A - KHA)^T + (I - KH)Q(I - KH)^T + KRK^T$$

Where $P_k = E(\tilde{X}_k\tilde{X}_k^T), \tilde{X_{k+1}} = \hat{X_{k+1}} - X_{k+1}, Q = E(ww^T), R = E(vv^T)$, Q is the system noise variance and R is the measurement noise variance.

Defining new variables L and M, $L = AP_kA^T + Q, M = HLH^T + R$ and continuing the expansion, we eventually find that the cost function is:

$$(K - LH^TM^{-1})M(K - LH^TM^{-1})^T - LH^TM^{-1}HL^T + L)$$

Since M is Positive definite, the best choice of K must eliminate the first dominating quadratic term.

$$K = LH^TM^{-1} = (AP_kA^T + Q)H^T(H(AP_kA^T + Q)H^T + R)^{-1}$$

The cost was originally defined to be the covariance of estimation: with new optimal value of K,

$$COST = P_{k+1} = E(\tilde{X_{k+1}}\tilde{X_{k+1}}^T) = L - KHL, \tilde{X_{k+1}} = \hat{X_{k+1}} - X_{k+1}.$$

Thus we derived the optimal gain K, estimation resulting estimation cov. P(k+1), posterior estimation X(k+1) and have what we need for the next round of recursive kalman filter update.

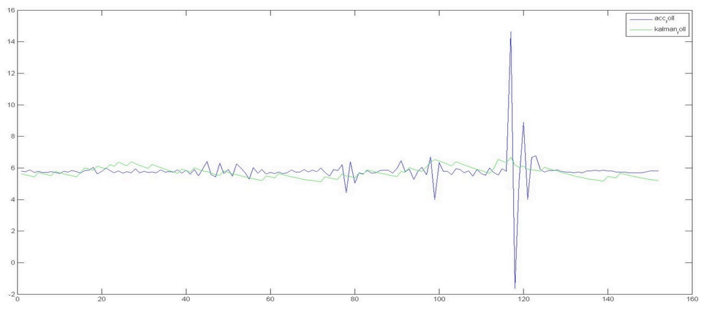
In our implementation of the filter, the pitch and roll angles were the state space vector X. With slight modification, our state transition equation became $X_{k+1} = AX_k + BU_k + w$ with control inputs. Here, U(k) will be our 3axis gyro inputs (these are angular rates) that drift but are relatively accurate in small time increment (hence an excellent choice for U(t):"control"). What was originally the system process noise would then becomes the Gaussian noise of the gyros.

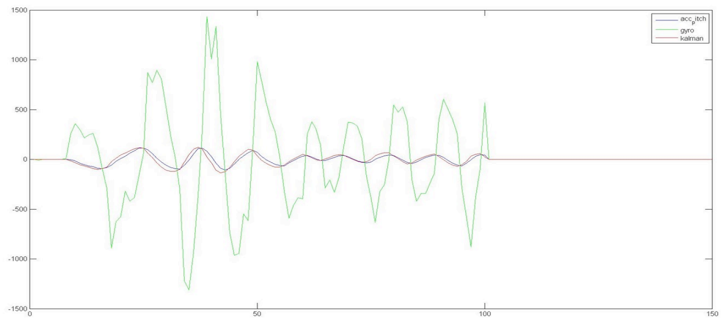
Once the priori propagation of the state and covariance has been made, H*X(t) (our prediction of what angle measurements are going to be, according to our prediction) is compared against the measurement of attitude calculated from the 3-axis accelerometer. This decision was made since the accelerometer is very noisy in small time scale compared to the gyro but its Gaussian noise is bounded unlike the gyros. The Kalman gain will reflect the propagation and the residual error (measurement-prediction) accordingly to update the state (pitch, yaw, and roll):

$$\hat{X}_{k+1} = A\hat{X}_k + K(accelerometerobservation - H(AX^k + BU_k))$$

Then R is our covariance matrix for the accelerometer's Gaussian noise. The constants R and Q are tuned by series of experiments recording the launch vibration of the rocket.

The result was very satisfactory. The filtered estimations were insensitive to launch shocks (2nd figure) while being sensitive to real rocket rotations (1st figure, red is the kalman filtered roll, blue is the actual angle measured: they are closely following each other). We now have reliable means to prepare attitude reference for MDP update.





IV. Rocket Guidance Algorithm

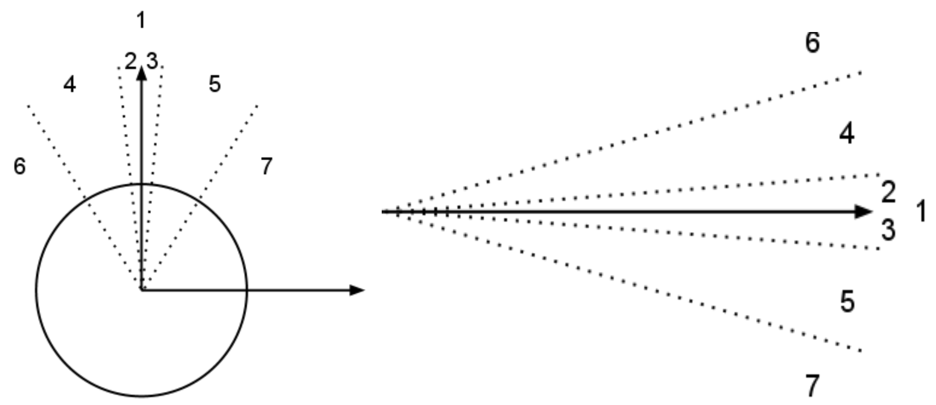
We implemented an reinforcement learning based guidance system that controlled the pitch, roll, and direction of the rocket. For proof of concept and due to technical limitations in our project, we wrote a separate MDP algorithm to control each of these features individually, solving them through value iteration. Given more time and resources, these three algorithms could be combined into a single MDP that would yield a comprehensive guiding policy.[1] Alternately, we could seek to formulate a model for linearly combining the results of the three separate MDPs to best reach targets. The state discretization and transition actions we used are described below.

[1] Several students who stopped by our project pointed out that a wing-leveller serves nearly the same purpose as our current pitch and roll stabilizing reinforcement learning algorithms. While this is true, we maintain that

Modeling States:

In modeling states, we ran into a fundamental difficulty: without an accurate simulator, we were unable to generate enough flight data to learn a comprehensive flight guidance policy, and given the time limitations of the project, building a simulation from scratch was beyond our means. Though we could technically include in our states the pitch, yaw, roll, GPS coordinates, velocity, and angular velocity in the three dimensions, the exponential growth of the number of states would make it impossible for us to learn the probability transition matrix P, since we were limited in learning these parameters through flight[2]. Thus a linear regression of states, though mathematically rigorous, would in fact serve us poorly; we would never be able to observe enough transitions to derive an accurate P, thus threatening the validity learned values and actions.

Due to this difficulty, we decided to first focus our efforts on stable flight, since during our test launches we found that the rocket would always spin randomly due to aerodynamic instabilities in its design. We divided states based solely on pitch, and then on roll.



Shown to the left is a visual representation of the way we discretized the roll and pitch of the rocket. We divided each of these up into seven different states, with a reward matrix of [2, 1, 1, 0, 0, -1. -1] corresponding to states 1-7. For pitch, we dynamically adjusted the rewards based on distance to a target location - initially the reward matrix was weighted highest at states 6 and 4, then as the rocket gets about halfway to the target, states 1, 2, 3 have the highest rewards, and finally as the rocket gets close to the target states 5 and 7 receive the highest weights.

While theoretically the servo motors can turn to any angle from 1-180, we had to limit the transitional actions to as few as possible to facilitate efficient learning. For pitch, we had the actions “up”, “flat”, and “down”, reflecting the overall angle of the wings. For example, at “up”, both wings were angled upwards at 20 degrees. Similarly, for pitch, we had the actions “left”, “right” and “flat”, signifying the direction in which the rocket should roll given the wing angles. For direction, “left”, “right” and “flat” were used again, but now only the outside fin was turned to the appropriate direction if necessary.

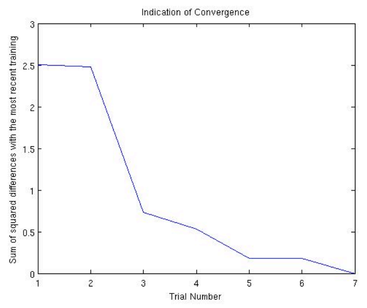
V. Results

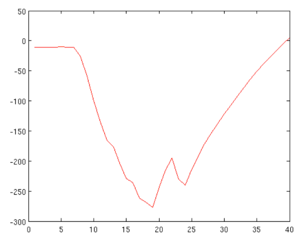
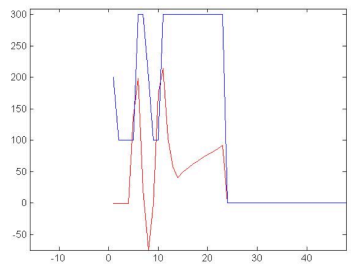
We found that control of the pitch over time, while useful, would be hard to test and yield uninteresting results, since gravity determines a great deal of the rocket’s pitch in a flight. However, we

reinforcement learning as a complete model is more accurately suited to rocket flight, especially when considering the variety of other tricks we could consider having the rocket do.

[2] We had not only time limitations, but technological limitations. Each flight threatened the life of our rocket; the lifespan of the rocket is necessarily limited by the number of landing impacts it can take.

demonstrated significant improvement in the rocket's roll during a launch. Shown on the left below is a graph of our change in transitional probabilities (P) over successive launches. The decreasing in difference is encouraging, as it suggests convergence. In the middle, we have a graph of roll versus time without controls. We can see that it is relatively random. On the far right, we have a graph of roll versus time with controls (in red), and the actions (in blue) that the rocket performed. A value of 100 signifies turning clockwise, a value of 200 signifies staying flat, and a value of 300 signifies turning counterclockwise. We can see that when the red line rises above zero, the blue line quickly shoots to 300, showing that the rocket seeks to level itself. When the red line falls below zero, the blue line shoots to 100, showing that the rocket is orienting itself in the opposite direction. This is exactly as desired.



VI. Further work

During this project we sent readings to computer running MATLAB to do computations. By doing calculations directly on the flight computer, we could improve the action loop rate, speeding up our reaction time to state changes.

With more time and a simulator, we could consider more states and more actions. Instead of set angles to turn wings, we could discretize the possible angles, and test out many more possibilities. Including variables such as velocity and angular momentum would make our states more accurate.

Finally, a linear combination of the instructions outputted by each of our three MDPs at a given state would allow the rocket to reach a target while maintaining stability. Learning these weights, or redoing the MDP process with the states of each MDP combined achieves the broad goal we had in mind when beginning this project.