FINAL REPORT: *Hot Wheelz* (Wednesday Lab 3 - 5 PM)

Yerbol Aussat, Kevin Chen, Scott Hallock, Jackson Hendershott,
Lavanya Jawaharlal, Albert Lai, Stephen Whiting

ABSTRACT

The Balance Car is a free-rolling car with a tilting balance platform mounted on top. The balance platform, controlled by a servo-motor and microcontroller, responds to both the tilt (pitch) and the longitudinal acceleration of the car so that a small object will be cradled on the platform as the car rolls down the roll-car track. A 9 degree-of-freedom inertial measurement unit (IMU) mounted on the car sends angular velocity, acceleration, and heading data to the micro-controller, which is used to calculate the pitch of the car. We applied three different filters - first-order complementary, second-order complementary, and Kalman - to the IMU data to improve the performance of the balance platform. We evaluated performance of each filter qualitatively via visual inspection, and also with a simple pass/fail experiment. The Kalman filter, with a time constant of 15ms, was the overall best-performing filter for our application.

INTRODUCTION

Based on our shared fascination with Hot Wheels cars, our team embarked on a journey of discovery towards creating a great roll-car project. Our project included the design of a roll-car with a balance platform, and then an experiment to compare the efficacy of different filters. The controls problem of self-balancing and self-orientating platforms is ubiquitous, with applications in the aerospace, automotive, and consumer goods industry. Some examples include the Segway – a self-balancing vehicle, the camera gimbal which stabilizes a camera, and even military tanks like the German Leopard 2 which include a mechanism to stabilize the gun barrel. Our project and subsequent experimentation ultimately aimed to enlighten our group on the concepts of data gathering, data filtering, and fine tuning of both vehicle design and control.

After finalizing the project specifications and parameters, we aimed to build a successful car platform that would adjust to the rolling car's change in pitch angle as it traversed down the track. This main objective was composed of many smaller objectives that included successfully implementing a controller for the servo motor and successfully balancing an object on the platform actuated by the servo motor. We decided to implement an open loop controls system as opposed to a closed-loop system because after very prototype testing, we decided that an open-loop system with additional data processing would be sufficient to accomplish our goals.

For the remainder of the project, our team divided into two sub-teams consisting of a Mechanical Design Team and a Controls Team. The Mechanical Team implemented a chassis to house the electronics that was both compact and efficient for traversing the track while the Controls Team created an effective controller for data processing and actuating the platform.

Figure 1: *LSM9DS1 inertial measurement unit (IMU)*

For our data capturing purposes we decided to use the LSM9DS1 9 degrees-of-freedom IMU for sensing the pitch angle of the car. The 9 degrees of freedom include a 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer, so the IMU can measure three important properties of the movement -

acceleration, angular velocity, and heading. However, for our application, we were only concerned with the accelerometer and gyroscope readings.

The angle of tilt can be determined by using the measurements from accelerometer. It is possible because the force of gravity acting on the sensor creates a constant acceleration pointing downwards. When the sensor not orthogonal to the gravitational field, the gravitational acceleration splits into x-, y-, and z-components from which the pitch, roll, and yaw can be calculated.

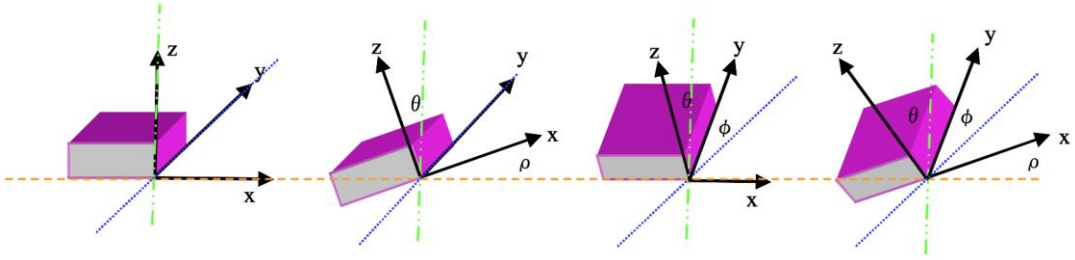

Figure 2: *Three Axes of IMU used to measure tilt.*

For our project we were primarily interested in the pitch measurements, which can be found with the following equation, using readings from the accelerometer.

$$\rho = \arctan\left(\frac{A_X}{\sqrt{A_Y^2 + A_Z^2}}\right)$$

The problem of using only the accelerometer for measuring the pitch is that this method is reliable only for the cases when gravitational acceleration is the only acceleration acting on the sensor. In our application, the car was linearly accelerating along the track (in x and z directions), which would affect the pitch values estimated by the formula above, making them inaccurate and very noisy. That is why this approach cannot be considered reliable.

A gyroscope is unaffected by accelerations, and always gives the measurement of the rate of angular change. Therefore, the tilt angle could be derived very accurately by integrating a gyroscope reading over time. The problem with using a gyroscope is that integration over time is inherently unstable because of the accumulation of intrinsic errors in measurement. This causes the gyro to drift from the correct value. Therefore, the gyroscope is reliable only for short durations, while the accelerometer produces noisy data but the mean of this data is relatively close to the correct value. A compromise can be made by combining the two sensors to get the best angle estimates while reducing the sensor noise. This is most commonly achieved using a Complementary filter or a Kalman filter.

METHODS

The goal of our experiment was to carry a die on our Balance Car's balance-platform without dropping it while the car rolls freely down the track. To start off, the Controls Team settled on a 9 degree-of-freedom IMU, a servo motor to position the platform, and an Arduino UNO microcontroller to interpret measurements from our sensor and send signals to our servo. Upon receiving these component parts and dimensions, the Mechanical Team dove right into designing the chassis and hardware.

Mechanical Design:

The Mechanical Team started with a balance-platform design that included two holes to mate with the servo-arm, as shown in *Figure 3*. After successfully 3D-printing the platform, we moved on to the chassis design. In order for the car to fit well on the track, the wheels would need to span the width of the square guides. This width, which was determined to be 2.5 inches, would be the minimum distance between our wheels. The chassis needed to include all of the controls, wiring, and battery pack. The largest electrical components were the Arduino Uno and battery pack, which meant we had to be careful and deliberate about the length and width of the acrylic mounting board. After some discussion, we decided that 3D-printing our chassis would allow us to fit our components while maintaining a sleek aesthetic. In the spirit of Hot Wheels toy cars, we wanted to make our car look realistic and flashy. Ultimately, we decided to base our car off a Lamborghini. We were able to find open-source surface models of a Lamborghini at *grabcad.com*, but modifications were required to allow us to 3D print it. The model was actual size, so we scaled it down by a factor of 20. We further modified the model by creating solid blocks and carving them into the shape of the front and rear of the car. We decided that to make the wheels roll smoothly, we would use press-fit skateboard bearings in our car body.

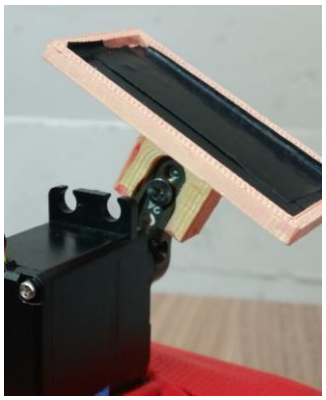

Figure 3: *Balance-platform mounted to servo-arm.*

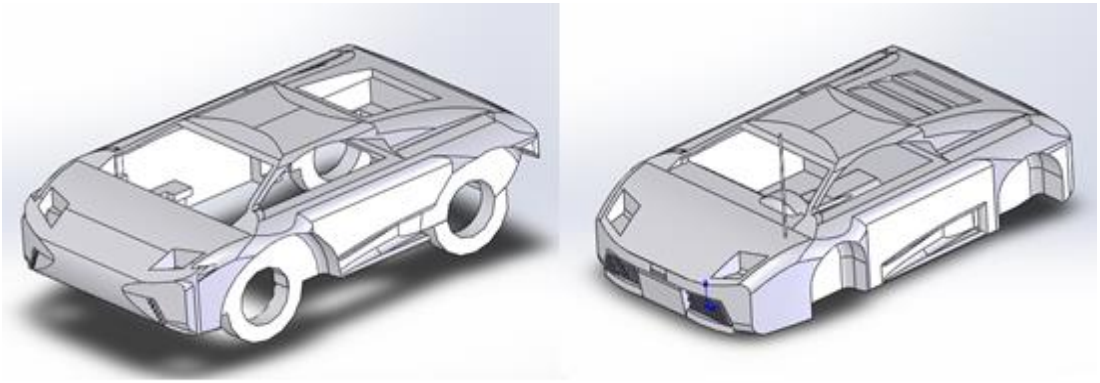

Figure 4: *Modified car body suitable for 3D-printing (left), and original Lamborghini model (right).*

3D-printed parts are fairly expensive - $5 per cubic inch, and support material is similarly priced. To minimize costs we removed excess material on the inside of our body while leaving approximately a quarter inch thick surface. Finally, we added a slot for our electronics board to easily slide into. Our chassis used 7.53 cubic inches of ABS and 5.2 cubic inches of dissolvable support. It took 14 hours to print and cost $90. With the Lamborghini design almost complete, the final step was to design wheels.

When designing our wheels we were mostly concerned with aesthetics. We decided to waterjet our wheels, which would allow us to make intricate shapes that could still be quickly manufactured. We looked at several styles of first Audi rims, and then Lamborghini rims before settling creating our final design. The wheels were cut out of ¼" polycarbonate with an outer diameter of 2" and an inner diameter of 8mm, the same as the inner diameter of our skateboard bearings. An 8mm aluminum rod fit our wheels and bearings perfectly.

Figure 5: *Inspiration for wheel design (left), wheel solid-model (center), 3D-printed and painted wheel (right).*

To assemble our car, we fit an acrylic plate into the car, on which we mounted the electronic components. The microcontroller was mounted to the underside of the car for easy access to reprogram the controls.

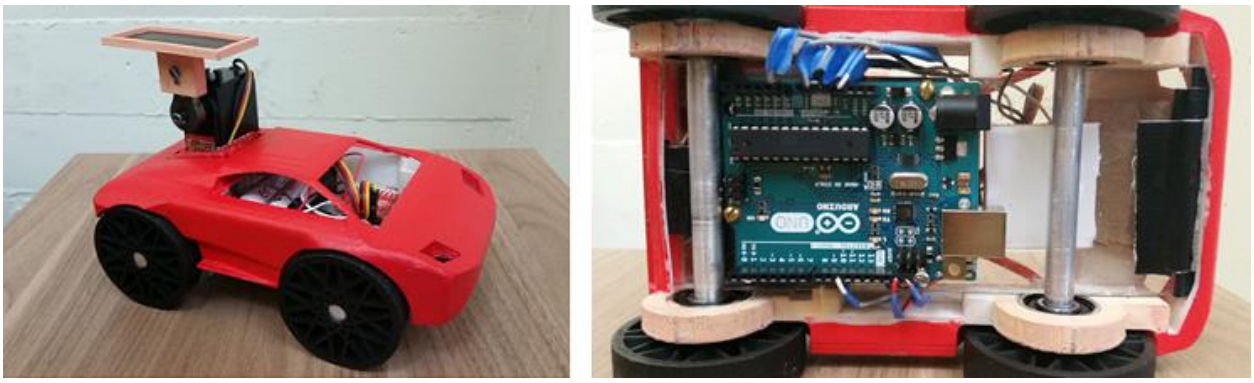

Figure 6: *Final assembly of Balance Car (left), underside of Balance Car showing Arduino UNO (right).*

Table 1: *Bill of Materials for the Balance Car.*

PART	**COST**
FDM Components: Chassis	$89.10
FDM Components: Platform	$ 5.90
Arduino	N/A - reused
LSM9DS1 IMU	$25
Polycarbonate for Wheels	$33
Servo motor	$25
Aluminum Rod (Car Axles)	$7
Rechargeable AA Batteries	$10
Total	**$195**

Controls:

While the mechanical team designed and created the chassis, the controls team experimented with different filters. As mentioned before, accelerometer data is susceptible to lateral and longitudinal acceleration influences in its calculations, and a common solution to this is to offset the readings with gyroscopic values. We approached the problem by first considering a first order complementary filter. This filter can be thought of as a coupled low pass filter for eliminating accelerometer noise and a high pass filter for eliminating gyroscopic drift.

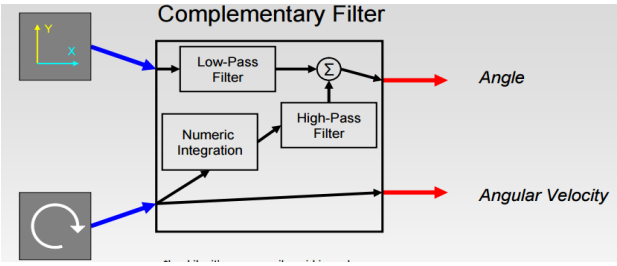

Figure 7: *Schematic for a Complementary Filter showing combination of high-pass and low-pass filter.*

This filter is implemented with the algorithm seen below and the sample code in the appendix.

$$Filtered\ Angle = \alpha\ \times (Gyroscope\ Angle) + (1 -\ \alpha)\ \times (Accelerometer\ Angle)$$

Alpha (α) is a coefficient that is chosen and fine-tuned by the team. Empirically, with our prototype, we found that the best alpha for us was 0.88. This number was determined partly from trial and error and partly through fine-tuning the time steps and time constant defined in our code. In addition to this complementary filter, we also implemented a second order complementary filter which is similar in principle to the first order algorithm, but has weights applied only to the change in accelerometer readings, and then offsets this value by the full value of the measured gyroscopic angle.

Finally, when we found through preliminary testing that both the first and second order filters only slightly reduced the noise found in the IMU readings, we decided to try the Kalman filter. The Kalman filter, also known as linear quadratic estimation, is a complex algorithm that makes estimates of future states of the system by using current measured values compared to previously measured values and predicting the most probable outcome. We were able to implement this with a fair amount of success using sample code from Robottini[3].

Test Trials:

For all of our test trials, we used a standard six-sided die as the object to balance, and we dropped the Balance Car from position 5 of the roll-car track. While we applied different filters to the servo for different trials, we used our Arduino to record the accelerometer data, both of the complementary filter data, and the Kalman filter data. Using this method, we were able to test each filter individually for different trials, and we also had comparative data for all of the filters for all of the trials.

The criteria for deciding which filter was the most advantageous for our application included seeing which filter had the ability to keep the die on the platform for the entirety of the car's motion. We also qualitatively analyzed the response time of the servo for each filter by

observing whether it exhibited lag or not and assessing its susceptibility to accelerometer noise by noting its jitteriness and stability when actuating. Finally, we were able to graph and compare representative results for each trial, some of which are included below. All of these criteria factored into the creation of the “Go/No-Go” table seen in the results section, showing whether or not a filter passed that specific trial. Trials in which the die fell off the balance platform were automatically considered to be a No-Go result.

RESULTS

Table 2: *Go/No-Go (pass/fail) test for different filters applied to our Balance Car.*

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
No Filter	NO GO	NO GO	GO	NO GO	NO GO
Comp 1	GO	NO GO	GO	GO	NO GO
Comp 2	NO GO	GO	GO	NO GO	GO
Kalman	GO	GO	GO	GO	GO

Trial 6	Trial 7	Trial 8	Trial 9	Trial 10
NO GO	NO GO	NO GO	NO GO	NO GO
NO GO	NO GO	NO GO	NO GO	GO
GO	NO GO	GO	GO	GO
GO	GO	GO	GO	GO

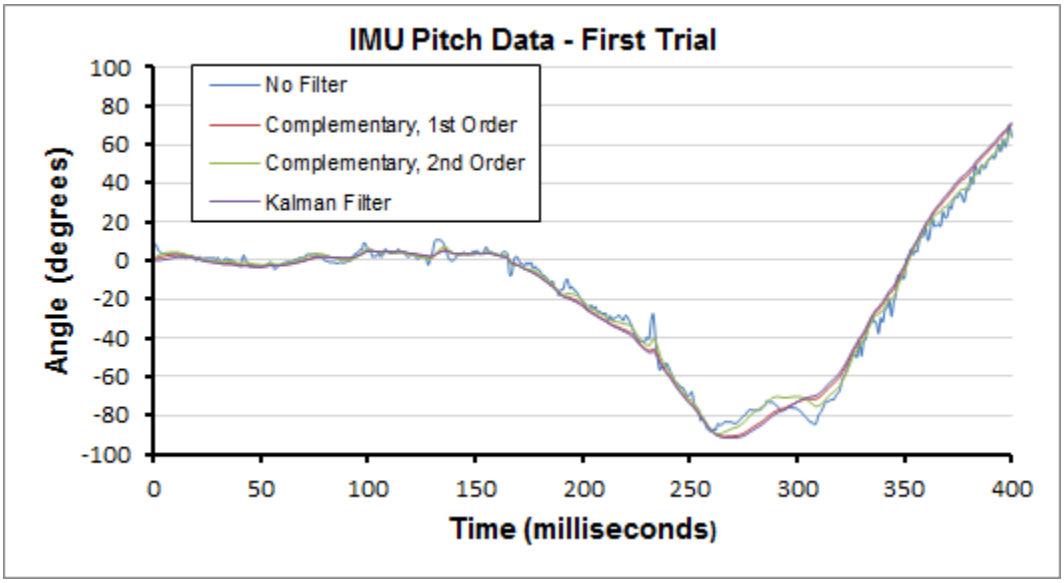

Figure 9: Comparison of no-filter, complementary first-order, complementary second-order, and Kalman filter applied to pitch angle data (calculated from IMU data). First trial.

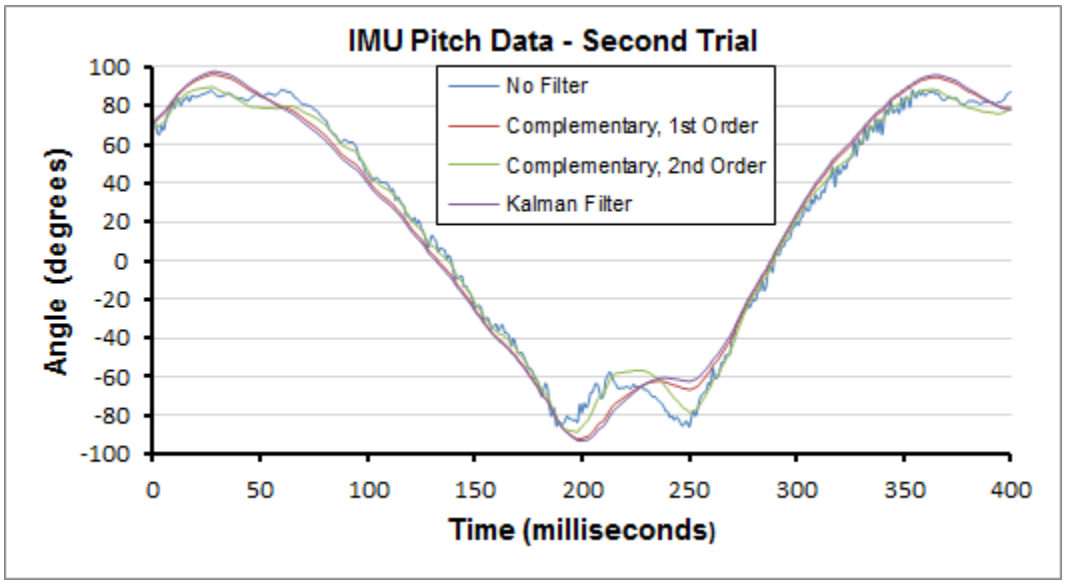

Figure 10: Comparison of no-filter, complementary first-order, complementary second-order, and Kalman filter applied to pitch angle data (calculated from IMU data). Second trial.

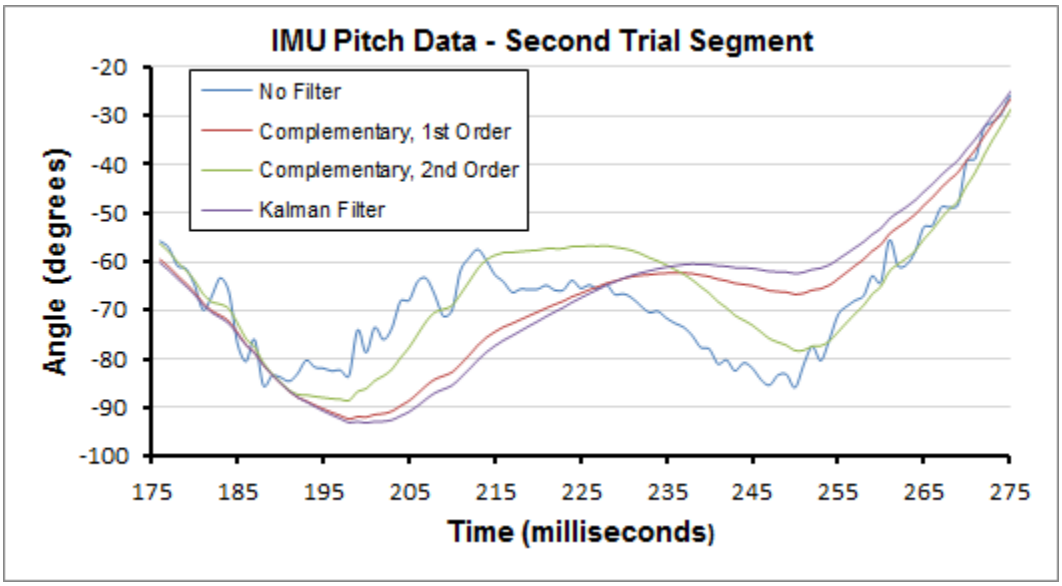

Figure 11: Close-up of 175ms-275ms domain of *Figure X* to compare filter behavior.

The results of our pass/fail test, shown in *Table 2*, illustrate that the complementary second order filter and the Kalman filter with a time constant of 15ms passed our criteria the most consistently with the first order complementary filter coming in at third. The Kalman filter was by far the most successful filter, with not a single failure in all 10 trials.

Data from *Figures 9* and *10* show that the filters behaved approximately the same for all the trials. All three filters were able smooth out the non-filtered accelerometer readings. The first-order complementary filter and the Kalman filter had similar response rates, but the Kalman filter dominates with respect to resistance to data noise. As shown in *Figure 11*, the Kalman filter and the first-order complementary filter are very similar with the exception that the Kalman filter has

more lag in its response time. What can also be seen in *Figure 11* is that the second order complementary filter has a much quicker response rate than either of the other two filters, and reacts to and follows change in acceleration very well. However, as a result of this, it lacks in its ability to suppress data noise.

DISCUSSION

The Balance Car worked exactly as planned, accomplishing both objectives to adapt to the angle of the track and acceleration of the car. Our controls team went through several filter iterations in order to find the optimal response time. This allowed the Balance Car to work accurately and consistently.

As we were integrating the chassis and controls, we ran into several space issues due to the battery pack and Arduino Uno. This was in part due to the mechanical design team streamlining the Lamborghini chassis as much as possible. However, we were eventually able to effectively fit all components onto the acrylic chassis. We ran into our biggest surprise during the initial testing phases. We quickly discovered that while the LSM9DS1 9-DOF sensor was perfect for our objectives, it created hurdles due to its high sensitivity. As the controls team worked to identify an effective filter, the mechanical team used small scraps of paper to properly level and zero the sensor.

Unfortunately we were not able to accomplish our stretch-goal of incorporating tri-color LEDs. For future improvements we could connect tri-color LEDs that respond to the acceleration of the car. For example, as the car slows down the headlights would turn red, and as it accelerates it would turn green. Other improvements to the Balance Car could include further optimization of the Kalman Filter or the addition of motors to power the car.

For future applications, the technology behind our Balance Car can be used in both passenger vehicles and shipping vehicles. Car seats could be fitted with a similar type of tilting mechanism to improve passenger comfort and safety over rough terrain or turbulence. Shipping vehicles such as ships and planes could benefit from the balance platform to assist in the transportation of fragile packages. We believe the most promising application is in the medical field: ambulances and medevac vehicles which transport passengers with injuries could be fitted with a balance-stretcher to stabilize patients en route to the hospital. This could be life-changing for patients with delicate neck or spinal-cord injuries.

CONCLUSIONS

The Balance Car was successfully able to carry a die down the entire length of the roll-car track. After deciding to use an open-loop system for balance-platform control, we implemented 3 different filter types - first-order complementary, second-order complementary, and Kalman - and evaluated the performance of each against a car without the filter. While the controls team implemented the filters, the mechanical team produced a car body and chassis, and designed the pass/fail experiment which we used to evaluate the filters. Each of the filters significantly improved the smoothness of actuation of the balance-platform, though to different degrees. The Kalman filter was the least susceptible to noise and longitudinal acceleration, though we had to adjust the time constant to find an appropriate balance between response time and smoothness. In the end, the Kalman filter with a time-constant of 15ms outperformed both of the complementary filters.

REFERENCES

[1] Colton, Shane. *Fun With the Complementary Filter. MIT.* *[http://scolton.blogspot.com/2012/09/fun-with-complementary-filter-multiwii.html]*

[2] Colton, Shane.*The Balance Filter.*MIT. *[https://b94be14129454da9cf7f056f5f8b89a9b17da0be.googledrive.com/host/0B0ZbiLZrqVa6Y2d3UjFVWDhNZms/filter.pdf]*

[3] Robottini. *Kalman Filter vs. Complementary Filter.* *[http://robottini.altervista.org/tag/complementary-filter]*

[4] Tuck, Kimberly. *Tilt Sensing Using Linear Accelerometers*. Accelerometer Systems and Applications Engineering. Tempe, AZ [http://www.thierry-lequeu.fr/data/AN3461.

APPENDIX

Our code was adapted from: *http://robottini.altervista.org/tag/complementary-filter*

The following is the Arduino code for the methods that:

- determine pitch angle using accelerometer
- apply the first-order complementary filter
- apply the second-order complementary filter
- apply the Kalman filter

```
// Method that calculates tilt angle using accelerometer only
float getPitchAcc(
float ax, float ay, float az)
{
  // Calculate raw pitch data using the accelerometer values
  float pitch = atan2(-ax, sqrt(ay * ay + az * az));
  pitch = pitch * 180.0 / PI;
  return pitch;
}

// Method that calculates tilt angle using first-order complementary filter
float Complementary1(
float newAngle, float newRate, float looptime)
{
  float dtC = float(looptime) / 1000.0;
  a = tau/(tau+dtC);
  x_angleC = a* (x_angleC + newRate*dtC) + (1-a) * (newAngle);
  //Serial.print("Complementary1: ");
  //Serial.println(x_angleC);
  //Serial.println("------------------------------");
  return x_angleC;
}

// Method that calculates tilt angle using second-order complementary filter
float Complementary2(float newAngle, float newRate,int looptime) {
float k=10;
float dtc2=float(looptime)/1000.0;

x1 = (newAngle -   x_angle2C)*k*k;
y1 = dtc2*x1 + y1;
x2 = y1 + (newAngle -   x_angle2C)*2*k + newRate;
x_angle2C = dtc2*x2 + x_angle2C;
return x_angle2C;
}

// Method that calculates tilt angle using Kalman filter
float kalmanCalculate(float newAngle, float newRate,int looptime)
{
```

```
float dt = float(looptime)/1000;
x_angleK += dt * (newRate - x_bias);
P_00 +=  - dt * (P_10 + P_01) + Q_angle * dt;
P_01 +=  - dt * P_11;
P_10 +=  - dt * P_11;
P_11 +=  + Q_gyro * dt;

y = newAngle - x_angleK;
S = P_00 + R_angle;
K_0 = P_00 / S;
K_1 = P_10 / S;

x_angleK +=  K_0 * y;
x_bias  +=  K_1 * y;
P_00 -= K_0 * P_00;
P_01 -= K_0 * P_01;
P_10 -= K_1 * P_00;
P_11 -= K_1 * P_01;

return x_angleK;
}
```