School of Electrical, Electronic, and Computer Engineering

Autonomous Vehicle Reliability and Localization

Manu Adina-Zada (21135495)

Supervisor: Professor Dr. Thomas Bräunl

Submitted:

Word Count:

Nomenclature

SAE	Society of Automotive Engineers
REV	Renewable Energy Vehicles
UWA	University of Western Australia
LiDAR	Light Detection And Ranging
GPS	Global Positioning System
IMU	Inertial Measurement Unit
EKF	Extended Kalman Filter
UKF	Unscented Kalman Filter
SLAM	Simultaneous Localization And Mapping
ROS	Robot Operating System
GNSS/INS	Global Navigation Satellite System / Inertial Navigation System
PCB	Printed Circuit Board
IC	Integrated Circuit

Abstract

Autonomous vehicles need to orientate themselves in the environment to be able to generate maneuverers to achieve a specific goal. Few sensors exist for this purpose, such as IMU, GPS and odometry. However, these sensors have different precisions and their own advantages and disadvantages. Hence, in the following dissertation different configurations of sensor fusion are going to be explored.

Since a standard Kalman filter can't be applied due to the non-linearity of the system model, the sensor fusion was performed by an extended Kalman filter (EKF). Since, odometry and IMU rely on past estimates to determine current position, these sensors become unreliable long term because of the accumulation of past errors. GPS on the other hand, doesn't require past estimates which makes it good for measuring the position long term. However, due to the discrete jumps and high variance of ±1m the estimations from the GPS are poor initially. After applying an EKF, the results have shown that the filter slightly improved the position estimate, due to utilizing the advantages of odometry and IMU.

Acknowledgements

I would like to thank the following people,

Dr. Thomas Braunl for providing very interesting and practical projects to the University of Western Australia. In addition, I thank Dr. Braunl for providing advice, vision and leadership.

REV SAE team for their teamwork and assistance throughout the year.

Family and friends for supporting me throughout the duration of this course.

Contents

1. Introduction

The autonomous vehicles are becoming more prevalent in our society. This is due to driverless cars having the potential to significantly reduce accidents, transit times, and reduce costs for industries (i.e. mining and taxi services). Companies such as Uber, Baidu, Google and Tesla have entered the research and development to attempt to produce a safe, fully autonomous and accessible vehicle.

The University of Western Australia have also entered this field in 2010. Initially the SAE vehicle has started as an electric vehicle and was developed into having autonomous capabilities over time. The vehicle in the current state has the required hardware for autonomous operation, however the software layer is insufficient and requires further development. Therefore, the focus of this thesis is on one of the lacking areas, which is the localization of the vehicle. Since the robot needs to know where it is in the environment, it needs to have an ability to track its position. Hence, a standard approach is to have sensors, such as GPS, IMU and odometry. Since sensors experience a lot of noise, to achieve the best precision, sensor fusion is performed which will be discussed further in this paper.

One of the other issues with the SAE vehicle was the reliability. Since a lot of hardware was made on prototype breadboards, the system was highly unreliable. One of the main culprits was the low-level circuit that's responsible for steering, breaking and acceleration. Hence, it's an important piece for the SAE vehicle to function. To improve the reliability, the author converted the low-level circuit into a PCB. The process will be further discussed in the later sections.

2. Literature Review

State estimation is one of the most important problems in autonomous navigation. Having an accurate state estimation is crucial for making optimal decisions for future control inputs to effectively navigate the environment. If the robot has a target destination, the robot needs to know it's current state which consists of position, velocity, acceleration and heading to correctly execute the right maneuvers to reach the goal. To get the current state, the robot is usually equipped with sensors, such as GPS, odometry and IMU. However, these are susceptible to noise and imperfections which introduce uncertainty to the measurements. Hence, the filters goal is to use all the available sensor data, as well as the robot's own dynamics to obtain a more precise estimate of the robot's state.

The first filter that was used for state estimation is the Kalman filter. The filter was introduced by R.E. Kalman in 1960 for linear systems with Gaussian process and measurement noise [1]. The Kalman filter ended up being a popular estimator, where it can be seen in aerospace and aircraft industries to seismology and weather forecasting [1]. Since the standard Kalman filter could only be applied for linear systems, a couple of variations of the Kalman filter were introduced to deal with non-linear systems [1]. One of the variations is the Extended Kalman filter, where it deals with non-linearity by approximating a linear equivalent before performing the required filtering sequence [1]. Since the Extended Kalman filter poorly approximates the linear equivalent for highly non-linear systems, a better approach was introduced. The new approach is the Unscented Kalman filter [1]. The UKF approximates the Gaussian equivalent of a non-Gaussian distribution and achieves better precision in comparison to EKF while having similar time complexity [1]. Both methods are better described in 2.1 and 2.2.

An alternative method for non-linear systems is the Markov Chain Monte Carlo filter or known as Particle filter [1]. The advantage of a particle filer is that it can be applied on systems with non-Gaussian distributions [1]. It functions by simulating the system evolution multiple times and choosing the state estimate as a weighted average of all simulations [1]. This has been applied to a lot of robotic applications, SLAM in particular [1]. Previously the particle filter wasn't as adopted as it now due to high computational cost. However, now due to the rise in computational power these filters are becoming more prevalent. In addition, more efficient variations of the particle filter such as Rao-Blackwellized particle filter has been developed that combines Kalman and particle approach [1].

In 2013, Thomas Drage has implemented a standard Kalman filter for estimating the position and Elmenreich algorithm for heading [2]. For position, he had a prediction step and used GPS and IMU for the correction step [2]. This is an adequate approach, however due to odometry being introduced by

Mitchell Poole in 2017 the position estimate could be further improved. In addition, the Kalman operated in 2D space, and a better estimate could be achieved using the 3D space. Additionally, since the orientation requires sine and cosine functions, it introduces non-linearity to the system. Hence, the original Kalman isn't sufficient. Furthermore, if a variation of a Kalman filter that accepts non-linearity is to be used, then the Elmenreich becomes redundant since the whole state (position and orientation) can be handled with one filter.

In 2013, Thomas Drage fused the heading using the following equation, by using the Elmenreich method,

$$Z = w_G X_G + w_I X_I$$

where the orientation, is a combination of two readings, one from GPS and the other is from IMU [2]. The w_G and w_I are weights that adjust how much influence the readings X_G and X_I have [2]. This method avoids the problem of non-linearity with the standard Kalman filter. However, since this method isn't using the prediction to improve the certainty of the orientation, this is an inferior method to the Kalman approach. Hence, as stated in the previous paragraph, to improve the certainty a Kalman type filter that accepts non-linearity should be used.

2.1. Kalman Filter

The Kalman Filter attempts to get the most optimal estimate with the data provided. The filter achieves this through recursive sets of actions [3]. The filter consists of two steps, the prediction and correction [3]. In the prediction stage, the Kalman filter attempts to predict the future state using the current state [3]. In the correction stage, the filter uses the measurements acquired to correct the prediction [3]. The magnitude of correction that is going to be performed is dependent on the uncertainty of measurement and the uncertainty of the prediction [3].

The Kalman Filter's prediction is denoted by the following expression [3],

$$x_k = A_k x_{k-1} + B_k u_k + w_k$$

where x_{k-1} is the previous estimate, u_k is the control matrix and w_k is the process noise [3]. The A is the state transition matrix, and B is the control input matrix [3].

The Kalman Filter's correction is denoted by the following [3],

$$x_k = \hat{x}_k + K_k(z - H_k \hat{x}_k)$$

where K is the Kalman gain that dictates how much of the correction is going to be performed, and z is the measurement [3]. The remaining term H_k is the matrix that transforms the predicted state into the measurement space [3].

The Kalman Filter relies on one of the properties of the Gaussian distribution [3]. The key property that the filter is taking advantage of is that when two or more Gaussian distributions are multiplied, the result will have a Gaussian distribution as well [3]. Hence, by using the Gaussian functions of two distributions, a new mean and variance can be calculated [3].

The new calculated mean and variance will then equal to [3],

Where:

$N(\mu_p, \sigma_p^2)$ predicted distribution

$N(\mu_m, \sigma_m^2)$ measurement distribution

$N_f(\mu_f, \sigma_f^2)$ filtered distribution

$$N_f(\mu_f, \sigma_f^2) = N(\mu_p, \sigma_p^2) * N(\mu_m, \sigma_m^2)$$

$$= \frac{1}{\sqrt{2\pi\sigma_p^2}} e^{-\frac{(x-\mu_p)^2}{2\sigma_p^2}} * \frac{1}{\sqrt{2\pi\sigma_m^2}} e^{-\frac{(x-\mu_m)^2}{2\sigma_m^2}}$$

$$\Rightarrow \mu_f = \mu_p + \left(\frac{H\sigma_p^2}{H\sigma_p^2 + \sigma_m^2}\right)(\mu_m - H\mu_p)$$

$$\Rightarrow \sigma_f^2 = \sigma_p^2 - \frac{\sigma_p^4}{\sigma_p^2 + \sigma_m^2}$$

Hence, the Kalman Gain is,

$$K = \frac{H\sigma_p^2}{H\sigma_p^2 + \sigma_m^2}$$

The Kalman gain determines how much of the correction is needed. If prediction is uncertain compared to measurement, then $\sigma_p^2 \gg \sigma_m^2$. This results in the Kalman gain to tend to go to 1 which means that more correction is going to be performed. However, when the opposite is true, where $\sigma_p^2 \ll \sigma_m^2$ the K will tend to go to 0. Hence, less correction is going to be performed due to uncertain measurement.

2.2 Extended Kalman Filter Model

The standard Kalman filter relies on the linear model. When a linear transformation is applied to a Gaussian distribution, the output retains Gaussian [4]. However, when a non-linear transformation is applied to a Gaussian distribution, the output becomes non-Gaussian as shown by Figure 1. Since the

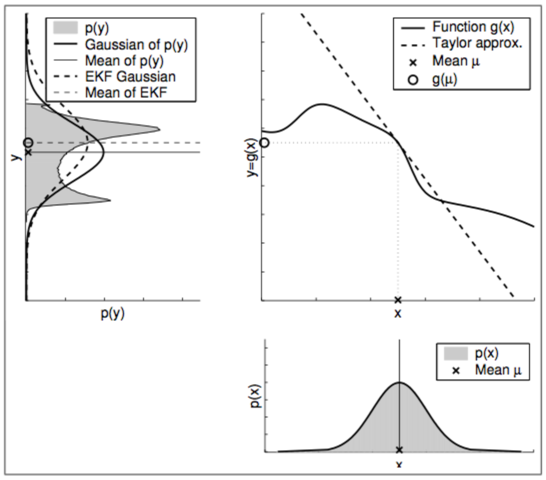

Figure 1, The Effect of a Non-Linear Transformation on the Gaussian Distribution

Kalman filter relies on the Gaussian distributions, if a model is non-linear then the filter can't perform its' function.

Since the computation of robotic movements require direction, sine and cosine functions are needed. These functions are non-linear, and as a result the standard Kalman filter can't be applied. The extended Kalman filter solves this problem by linearizing the non-linear function. To achieve this, the EKF uses first order Taylor series approximations [5].

The linear function is approximated by the following [6],

$$f(u_k, x_{k-1}) \approx f(u_k, \mu_{k-1}) + \frac{df(u_k, \mu_{k-1})}{dx_{k-1}}(x_{k-1} - \mu_{k-1})$$

Another local point μ_{k-1} is used to approximate a linear function around x_{k-1} [6]. The differential can be obtained from Jacobian of $f(u_k)$ [6]. The same principle is applied to the measurement model [6].

With an extended Kalman filter, the current pose is predicted using the past measurements [7]. As shown,

$$x_k = f(x_{k-1}) + w_{k-1}$$

where x_k is the current pose at time k, and the f is a non-linear transition function that converts the past state to the current state [7]. The w_{k-1} is the process noise that is normally distributed [7]. As of now, the SAE vehicle is using 2D coordinates. Hence the state x is composed of the x-coordinate, the y-coordinate and yaw. The state measurements that are being received are expressed as,

$$z_k = h(x_k) + v_k$$

where z_k is the measurement at time k [7]. The h is a non-linear function that converts the state into measurement space [7]. The last variable, v_k is the measurement noise that is normally distributed [7].

The first step of the EKF filter is to predict the next state and the next error covariance using the current state and current estimate error covariance [7]. The following two equations describe this,

$$x_k = f(x_{k-1})$$

$$P_k = FP_{k-1}F^T + Q$$

where f is composed of standard kinematics equations [7]. The matrix F is a Jacobian of f and P is the process error covariance [7]. The following sequence $FP_{k-1}F^T$ is to predict the future process error covariance using the current process error covariance using the Jacobian of f [7]. The remaining term Q is the process noise covariance [7].

The correction sequence that occurs for the EKF is identical to the standard Kalman filter. However, the linearized versions of f and h are used, and F and H are Jacobian of linearized f and h [7]. The following steps are [7],

1. Compute Kalman Gain,

$$K = P_k H^T (H P_k H^T + R)^{-1}$$

2. Perform the correction,

$$x_k = x_k + K(z - H x_k)$$

3. Calculate new process error,

$$P_k = (I - KH) P_k (I - KH)^T + KRK^T$$

3. Methodology

3.1 Robot Operating System (ROS)

In 2018 most of the software was scrapped or adapted to convert to ROS-based system. ROS greatly reduces the complexity of developing a software system for a robot. ROS provides low-level device control, implementation of commonly-used tools, message-passing between processes, and package management [8]. Hence, instead of creating an independent system where a ‘broker’ would manage the communication between ‘modules’ (programs that have a specific function) ROS readily provides these services. Hence, the user only has to worry about creating ‘nodes’ (programs that perform a certain function) that listen and talk to other nodes [8].

3.2. Sensors

3.2.1 IMU

A single sensor that has both capabilities of a GPS and IMU was used (Xsens MTi-G-710). To implement the sensor into the system, an available ROS Xsens driver package was used. The Xsens package works by receiving the sensor measurements and then inputting these readings into ROS messages. Other nodes can then subscribe to these messages to use the data given by that driver. Another node that will utilize these messages will be the ‘robot_localization’ package.

When it came to the covariance matrix of the IMU, the Xsens ROS driver used the default values specified by the manual.

3.2.2. GPS

The GPS functionality of the driver didn’t add the covariance to the GPS type message. Since, the Xsens Mti-G-710 has the capability of estimating its’ own horizontal and vertical accuracy this information just needed to be passed to the node. Hence, the vertical and horizontal accuracy was simply added to the ROS GPS type message. Depending on the environment, the accuracy would rise indicating poor precision if there were a lot of obstacles surrounding the sensor, and the accuracy would fall if the area was clear indicating better precision.

3.2.3. Odometry

The error was empirically estimated to be around 0.04 per meter. To calculate the error in terms of x and y coordinates, the error had to go through the Ackermann model. The equations for the model are as follows,

Where:

d_{left} is the distance the left wheel has travelled since the last reading

d_{right} is the distance the right wheel has travelled since the last reading

$d_{vehicle}$ the distance travelled by the vehicle since the last reading

Initially, the distance the vehicle has travelled is determined,

$$d_{vehicle} = \frac{d_{left} + d_{left}}{2}$$

To get the velocity of the vehicle,

$$v_{vehicle} = \frac{d_{vehicle}}{\Delta t}$$

x and y components of velocity can then be found by using the steering angle given by the servo,

$$v_x = v_{vehicle} * \cos\varphi$$

$$v_y = v_{vehicle} * \sin\varphi$$

Thus, the x and y coordinates can be determined,

$$x_n = x_{n-1} + v_x \Delta t$$

$$y_n = y_{n-1} + v_y \Delta t$$

Hence, the error should accumulate over time. However, the covariance values for x and y didn't get passed through to the Jetson from the low-level system due to an error in communication between these systems. Therefore, the covariance values for x and y were put as a static in the control program. This resulted in poor results, this is further discussed in the Results section.

3.3 Extended Kalman Filter Integration

For sensor fusion, the 'robot_localization' ROS package was used. This package provides a node that contains an EKF algorithm. In addition, the package can support multiple sensor and is able to transform GPS data for fusion. The advantage of using this package is that it handles all the sensor messages in the background. This involves syncing up the messages that are arriving at different times and ensuring that the EKF still runs when a sudden abruption is caused (i.e. sensors is damaged). The package can work in 3D or 2D mode.

For ROS packages, the nodes can be reconfigured once it's ran or it can be configured in the 'param' file. The 'param' file was set up so that the EKF is performed at 20 Hz.

For each sensor, the inputs need to be specified. The package provides a matrix,

$$\begin{pmatrix} x & y & z \\ \theta & \varphi & \psi \\ v_x & v_y & v_z \\ \dot{\theta} & \dot{\varphi} & \dot{\psi} \\ a_x & a_y & a_z \end{pmatrix}$$

Where x is an axis for forward and backward movements, y is an axis that spans left and right, and z axis is the vertical axis up and down. The symbol θ represents rotation around the x axis, φ is rotation around y, and ψ is the rotation around z. For the package, to enable each sensor input, each position in the matrix needs to be marked as true or false.

The odometry was configured as follows,

$$\begin{pmatrix} T & T & F \\ F & F & F \\ F & F & F \\ F & F & F \\ F & F & F \end{pmatrix}$$

Where T is true and F is false. The odometry is limited, as it can only provide x, y and ψ. The issue with our system is that yaw is taken from the commands that are sent to the steering servo. This is suboptimal, since the steering command that is sent to low level is oftentimes different than the steering that occurs in reality. Therefore, the covariance value for yaw is assigned as a high value and it is also disabled in the param file.

The IMU can provide all components of velocities and accelerations. In addition, it also provides all the rotations θ, φ and ψ. Hence, the configuration was set as the follows,

$$\begin{pmatrix} F & F & F \\ T & T & T \\ F & F & F \\ T & T & T \\ T & T & T \end{pmatrix}$$

The GPS is able to only provide x, y and z coordinates. The corresponding fields are marked as true.

$$\begin{pmatrix} T & T & T \\ F & F & F \\ F & F & F \\ F & F & F \\ F & F & F \end{pmatrix}$$

IMU and odometry provide continuous data, and by ROS convention these sensors operate in 'odom frame'. In odom frame, the pose (position and heading) can drift over time and the pose can only change smoothly. This means that there should be no discrete jumps. Hence, the IMU and odometry fusion is run on one node. Another node is then run simultaneously for discontinuous data such as GPS. The GPS operates in 'map frame' where drift is absent and discrete jumps are expected in-between readings.

3.4 PCB

The PCB was previously created on a breadboard. Due to the exposed wiring and poor soldering, this resulted in numerous faults occurring in a period of two months. When each fault occurred, the repair would take a long time due to the entangled wiring and large amounts of solder joining multiple lanes and elements together as shown by Figure 2. Therefore, the aim for the PCB was to improve reliability as well as to add an additional noise protection to the low-level circuit.

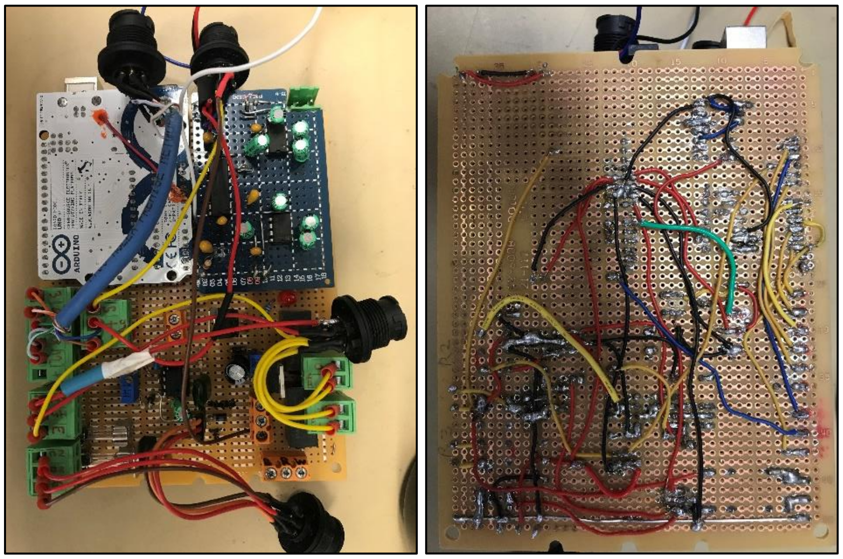

Figure 2, The state of low level before PCB (top layer on the left, and bottom layer on the right)

The first step was to trace the wiring and solder to create a schematic of the low-level circuit. The resulting schematic is shown in appendix A. The EAGLE software was used due to it being capable of both, the schematic and PCB creation.

The created schematic was then used to create the PCB. To reduce complexity, the number of layers was kept to 2, the top side and the bottom side. The PCB dimensions were needed to be identical to the previous breadboard circuit, so it could fit in the existing enclosure. Hence, the only parameter that could be controlled was the number of layers, and the advantage of having 2 layers is that it's cheap.

For all the connections between components, the traces were routed in paths that have shortest distances. Similarly, the components were arranged in a way to reduce the distances between each connected component as well. It's generally preferred to keep the trace length short to avoid the trace behaving like a transmission line [9]. When the length of the trace is in range of the signal's wavelength, the trace obtains the problems associated with the transmission line, such as time delay, reflections, and crosstalk [9]. Since the signal passes through the trace at finite speed, the signal takes some time to reach the destination [9]. Hence, there's more propagation delay if the trace is longer. In addition, the transmission lines are prone to reflections. If the impedance changes in the signal chain (e.g. source – trace – component) then the reflections will occur that cause overshoots or undershoots [9]. To avoid the reflections, the impedances need to match the characteristic impedance of the trace [9]. Another issue associated with transmission lines is crosstalk. When two traces are in parallel they may influence each other [9]. Due to the electromagnetic field, one trace can be influenced by another trace by an inductive and capacitive coupling [9]. Hence, a general rule is to keep the traces apart by a distance of twice the trace width [9]. This rule was followed when creating the low-level PCB. Overall, to avoid the issues associated with the transmission line, it's best to keep the trace lengths short.

When designing the board, creating vias was avoided. Vias are used to connect layers together. They are made by drilling a hole and pouring copper into the hole, as shown by Figure 3. The vias have increased

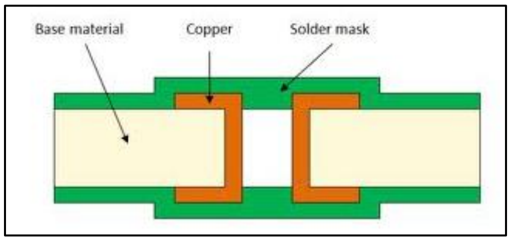

Figure 3, Via

impedances compared to traces, hence they are prone to creating reflections [9]. When creating the low-level PCB, the vias were avoided but in the expense of trace length, as shown in appendix A.

To further reduce radiation, the bend of the trace and trace positions were also considered. When the trace bends by a 90 degree angle, the capacitance increases in the region of the corner. The change in

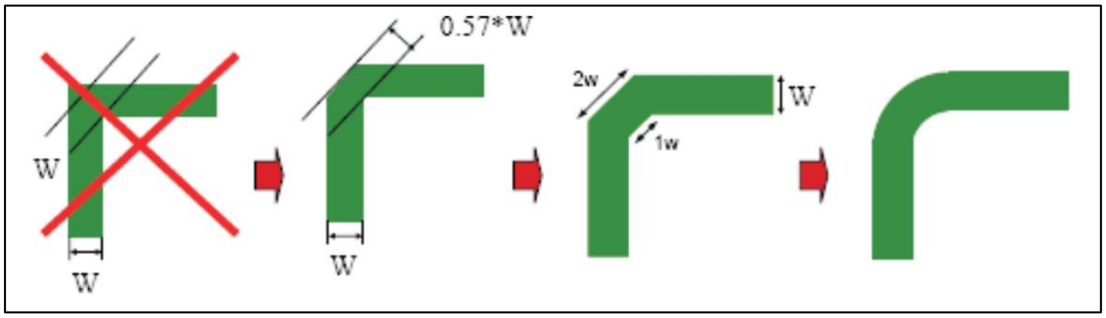

Figure 4, Optimal trace bend [9]

the impedance causes reflections [9]. Thus, it's best to have two 45 degree bends instead, as shown by Figure 4 [9]. This was kept in mind when designing the PCB, as shown in Appendix A, where 90 degree bends are absent. The crosstalk can also occur between adjacent layers if the traces in both layers are in a similar position and are in parallel [9]. Hence, it's better to place two traces orthogonally if the position can't be changed [9]. In the designed low-level PCB, this isn't an issue since most traces aren't in parallel.

Most ICs are prone to noise or ripples in the supply pins that in turn cause performance degradation [10]. For power supplies or the ICs, the decoupling capacitors are utilized as an effective way to reduce the noise when a steady DC voltage is needed [10]. The decoupling capacitors are used because they oppose quick changes in voltage [11]. Therefore, when the voltage spikes the capacitor absorbs the excess voltage and when the voltage drops the capacitor supplies the energy required to keep the voltage stable [11]. The designed low-level PCB has the decoupling capacitor (C23) at the isolation side of the circuit, as shown in Appendix A. The other PCB that was designed for odometry circuit has a decoupling capacitor for each IC and input power pins, also shown in Appendix A.

For both the low-level PCB and odometry PCB, the majority of ICs were chosen to be the through-hole instead of the surface mount. The through-hole components are generally larger and to solder them to the board, the pins have to go through a hole [12]. The surface-mount components on the other hand, are much smaller and are soldered on the surface of the board [12]. Since, the surface-mount components require experience to solder and are oftentimes more expensive, the through-hole components were chosen [12]. The added benefit of through-hole components is that they are more reliable, better secured to the board, and can withstand higher temperatures [12].

Another design decision was choosing the trace width and thickness. The trace width is decided by the current, thickness of copper and temperature [13]. When it comes to thickness, the manufacturers usually have two options, 1 oz or 2 oz/ft^2. Since the maximum current in the circuit is 0.5A, the trace width required for this much current is low. Since, there is no need to reduce the trace width further by choosing 2 oz, the 1 oz/ft^2 is reasonable. To determine the trace width, a graph provided by the

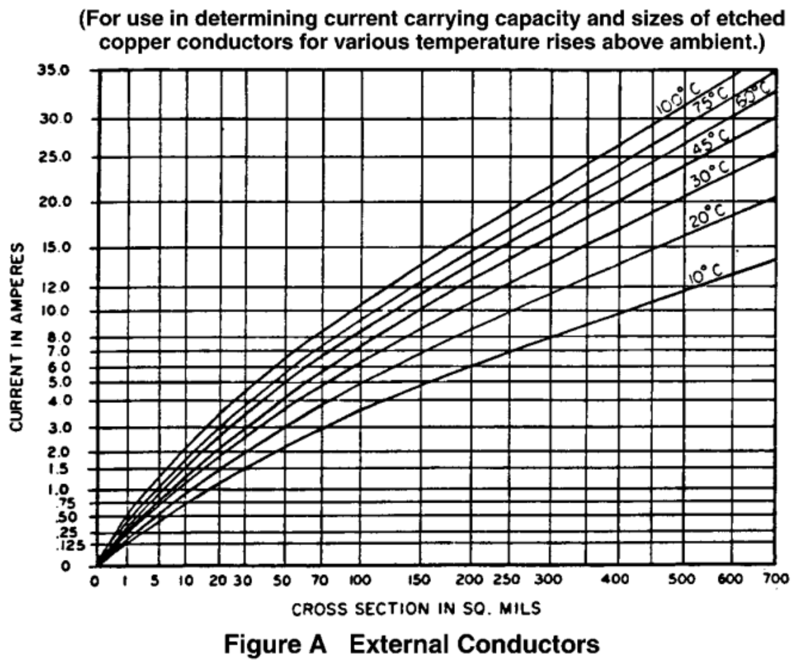

Figure 5, Conductor Thickness and Width for External Layers [13]

IPC2221A was used (or similarly a direct formula can be used) as shown Figure 5 [13]. Since there aren't any components that produce a lot of heat, a temperature rise of 10 degrees was expected. Hence, the trace width chosen was 10mil with thickness of 1oz/ft^2. The final PCB version that was printed and put together is shown in Appendix A.

4. Results

4.1 PCB

The low-level PCB was implemented to the SAE vehicle. The board didn't cause any major issues aside from minor problems that required slight adjustments. One such problem was that the Arduino Mega 2560 had a pin that didn't sink the current properly, and as a result, the connection was rerouted to another pin that was functional. Another issue was due to poor soldering on one of the relay pins. To solve the issue, the pin was re-soldered. Aside from those minor issues which were fixed, no other problems were encountered in four months since the PCB was implemented. Hence, in terms of the reliability, the PCB conversion was successful since no low-level faults occurred after several months of testing.

The odometry PCB was printed, but it still needs to be implemented due to inexperience with the surface-mount soldering. Soldering a microcontroller was proven to be difficult, hence the implementation will be done in the future with a person who's more experienced in this area. Once implemented, the PCB version of odometry should be much more reliable due to the use of decoupling capacitors. The old breadboard odometry circuit omitted any noise protection, so it is unreliable in comparison.

4.2 The Extended Kalman Filter

Due to the error in communication between the low-level and the Jetson, the odometry's covariance values didn't get through to the Jetson. Since, other variables were successfully received, the source of this error is most likely the low-level software. Even though attempts have been made, due to the time constraints, the error couldn't be fixed on time.

In the following table, the odometry measurements are compared to the fused odometry and IMU position estimate. To gather the data, the vehicle was driven in a relatively straight path. The ground that the vehicle was driven on didn't have any changes in elevation and as a result the Z coordinate was omitted. The data was gathered from three points in time.

Table 1: Data Taken at 50.88s

Position	**Odometry**	**Fused: Odometry + IMU**
X	1.42999	1.43639
Y	0.02999	0.02504
Covariance X	0.01600	0.01303
Covariance Y	0.01600	0.01344

Table 2: Data Taken at 100.91s

Position	**Odometry**	**Fused: Odometry + IMU**
X	41.0699	41.0727
Y	13.9399	13.9012
Covariance X	0.01600	0.01318
Covariance Y	0.01600	0.01317

Table 3: Data Taken at 150.89s

Position	**Odometry**	**Fused: Odometry + IMU**
X	58.0000	57.9703

Y	27.6800	27.6584
Covariance X	0.01600	0.01318
Covariance Y	0.01600	0.01326

As shown by the tables, due to static odometry's covariance values the filtered covariance values don't experience significant change. If the odometry functioned correctly, the expected covariance values of odometry and filtered data should've risen over time. The positive result that could be seen, is that the fused covariance values for X and Y are lower. This demonstrates that the filter is functioning as it was intended. Another observation is that there's slight adjustment made by the filter. This demonstrates that the adjustment isn't that significant in the short term. Hence, if odometry covariance data accumulated, it would be expected for the adjustment to be much more significant in the long term. Theoretically, due to the accumulation of errors by the IMU and odometry, the GPS is expected to offset the drift. However, as a result of the time constraints and issues associated with GPS, the fused GPS data couldn't be retrieved on time.

An EKF was already implemented by Tom Moore in 2015 using the same 'robot_localization' package on a differential drive robot. The results from this implementation better demonstrate the capabilities of this filter. The first figure shown below demonstrates the reference path in red (i.e. ground truth) and

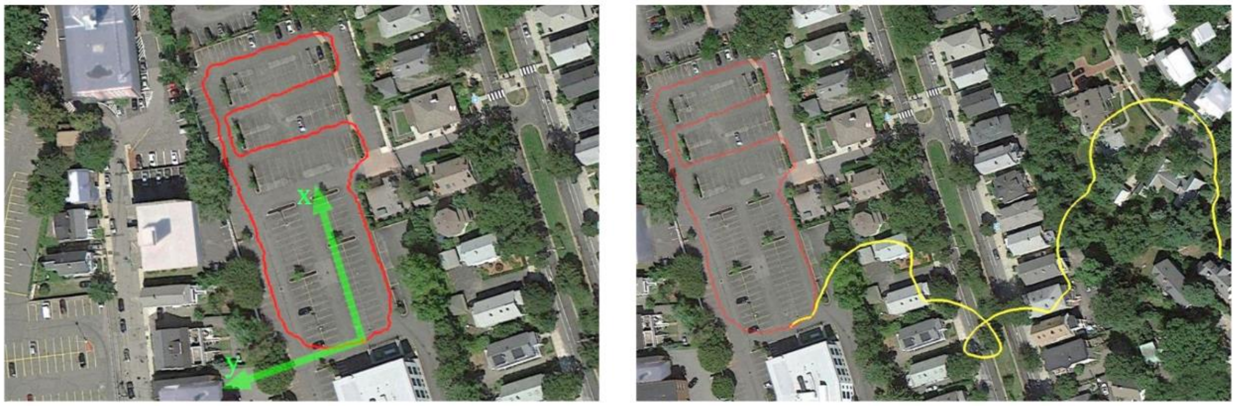

Figure 6, Left - Reference Path (Red); Right - Odometry Estimate (Yellow)

the odometry position estimate from odometry by yellow. As it can be seen from Figure 6, initially the odometry was accurate, but due to the buildup of error over time the position estimate and heading became very poor [7]. The next figure shows the effect of fusing the IMU + odometry on the left and fusing IMU + odometry + GPS on the right. As it can be seen in Figure 7, at the start IMU + odometry position estimates were fairly accurate, but due to accumulation of errors the estimate became imprecise long term [7]. However, in comparison to solely using odometry, the fusion with IMU greatly improved the estimate [7]. When GPS was introduced to the fusion of IMU and odometry, the Figure 7 demonstrates that the estimate is greatly improved long term [7]. Similar behavior was expected in this project's implementation despite having an Ackermann model. However, due to the problems that were described previously, these results weren't obtained.

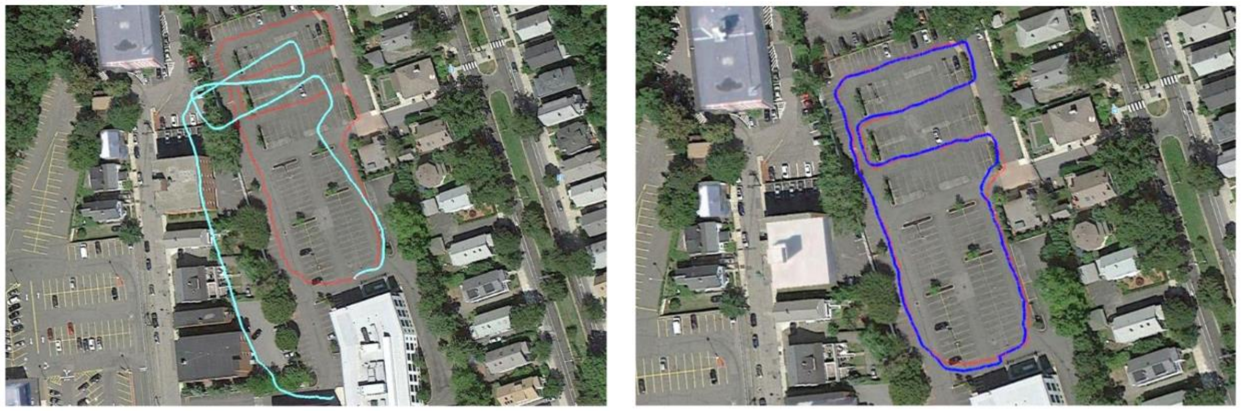
Figure 7, Left - IMU + Odometry (Cyan); Right - Odometry + IMU + GPS (Blue)

5. Future Work

The future work that could be done is the SLAM and the unscented Kalman filter. UKF solves the disadvantage of EKF and is generally considered as superior to EKF. SLAM further improves the position estimate and since the EKF is an important part of SLAM, it is also considered as future work.

5.1 Simultaneous Localization And Mapping (SLAM)

SLAM is an algorithm that is responsible for mapping an unknown environment while simultaneously navigating the environment using the generated map [14]. The whole purpose of SLAM is to use the environment to update robot's position [14]. The environment is mapped by vision (cameras), radar and LiDAR [14]. The generated map is used to track the positions of landmarks as the robot moves around [14]. The EKF is used extensively in SLAM [14]. The EKF is applied for position estimates of landmarks, as well as for the robot's position estimate [14]. The overview of SLAM is shown by Figure 8, however instead of having just odometry, the REV project has IMU and GPS.

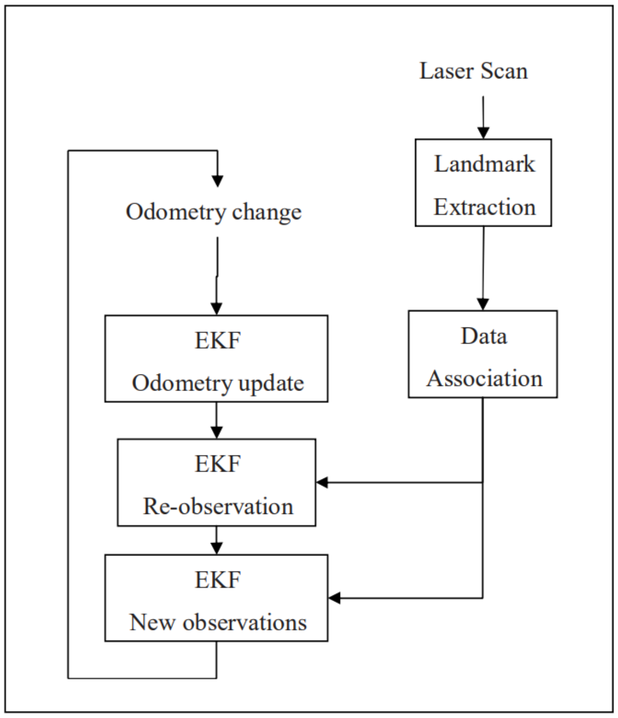

Figure 8, Overview of SLAM [14]

When the robot has moved, the robot receives a new position estimate from EKF through it's sensors (i.e. odometer, IMU, GPS) [14]. Then, the landmarks are extracted from the new observation of the environment when the robot is in the new position [14]. The robot then tries to associate the new landmarks to the landmarks previously seen [14]. If the robot identifies that it has re-observed some landmarks, it uses this information to update its position using the EKF [14].

SLAM will provide an additional source for estimating position. Especially, with a LiDAR sensor that provides an accurate data, the position estimate will further be improved. However, SLAM implementation is more complex and the cost of the equipment required is very high.

5.2 Unscented Kalman Filter (UKF)

When the model is highly non-linear, the linear approximation of EKF can be inaccurate. If the local point μ_k is used to generate the linear approximation, and the point at x_k is very far from μ_k due to high non-linearity, then the linear approximation becomes poor as shown by the Figure 10. The Unscented

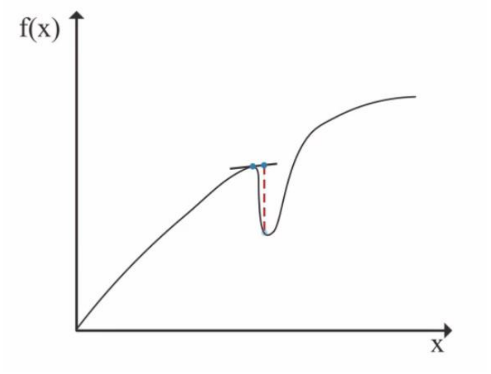

Figure 10, Poor Linear Approximation of the Non-Linear Function [16]

Kalman filter addresses the problem by deterministic sampling approach [15]. The state's gaussian distribution is first approximated by carefully chosen sampled points that completely capture the mean and variance of the distribution [15]. These sampled points are then transformed by the non-linear model and the output then represents a gaussian approximate of the non-gaussian distribution [15]. The

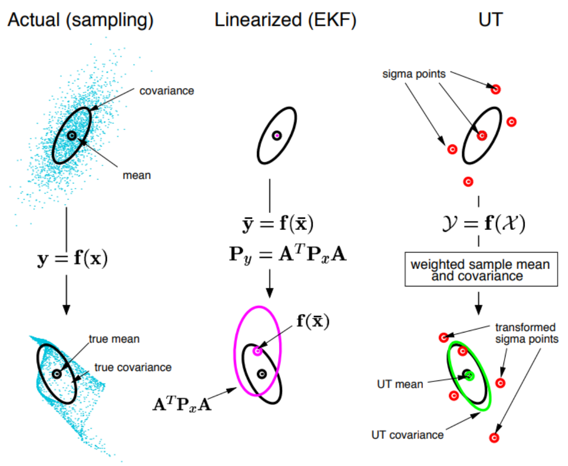
Actual (sampling)
Linearized (EKF)
UT
covariance
mean
sigma points
$\mathbf{y} = \mathbf{f}(\mathbf{x})$
$\bar{\mathbf{y}} = \mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{P}_y = \mathbf{A}^T\mathbf{P}_x\mathbf{A}$
$\mathcal{Y} = \mathbf{f}(\mathcal{X})$
weighted sample mean and covariance
true mean
true covariance
$\mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{A}^T\mathbf{P}_x\mathbf{A}$
transformed sigma points
UT mean
UT covariance

Figure 9, Different methods of dealing with Non-Linearity [15]

process is shown by right hand side of the Figure 9. It should be noted that the time complexity of UKF is equivalent to EKF, and the estimation by the UKF is accurate to the 3rd order of Taylor's series expansion [15]. Hence, the UKF is a superior approach for dealing with non-linearities.

6. Conclusion

The extended Kalman filter shows promise, however more work needs to be done on the implementation and extraction of the results. Due to the wide spread use of the EKF, the improvement in position estimate is expected. The results support this, where a slight improvement in covariance is seen. However, once implemented with GPS and with proper odometry covariance accumulation, the improvement in position estimate should be much greater than what it is currently shown. In the future, the position estimate can be further improved with UKF and SLAM.

The PCB greatly improved the reliability of the low-level circuit. Without the loose wiring and exposed chunks of solder, the causes behind numerous faults decreased. Since the design practices for reducing the transmission line effects were followed, the general performance of the circuit also improved. In addition, with the added noise protection, it is expected for the components of the circuit to last longer.

Bibliography

[1] R. Ivanov, "State Estimation Filters," Department of Computer and Infomation Science, Philadelphia.

[2] T. Drage, "Development of a Navigation Control System for," The University of Western Australia, Perth, 2013.

[3] R. Faragher, "Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation," *IEEE SIGNAL PROCESSING MAGAZINE,* vol. 29, no. 5, pp. 128-132, 2012.

[4] M. Taboga, "Lectures on Probability and Statistics," StatLect, 2010. [Online]. Available: https://www.statlect.com/probability-distributions/normal-distribution-linear-combinations. [Accessed 4 May 2018].

[5] M. Byron, K. V and S. Maneesh, "Derivation of Extended Kalman Filtering and Smoothing Equations," 2004.

[6] D. Morrell, "Extended Kalman Filter Lecture Notes," Arizona State University, Phoenix, 1997.

[7] T. Moore and D. Stouch, "A Generalized Extended Kalman Filter Implementation for the Robot Operating System," Springer, Massachusetts, 2014.

[8] "ROS Introduction," Open Source Robotics Foundation, [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Accessed 22 May 2018].

[9] A. Weiler, A. Pakosta and V. Ankur, "High-Speed Layout Guidelines," August 2017. [Online]. Available: http://www.ti.com/lit/an/scaa082a/scaa082a.pdf. [Accessed 26 May 2018].

[10] "Decoupling Techniques," Analogue Devices, 2009. [Online]. Available: http://www.analog.com/media/en/training-seminars/tutorials/MT-101.pdf?doc=CN0305.pdf. [Accessed 26 May 2018].

[11] "Coupling and Decoupling," capacitorguide.com, 2018. [Online]. Available: http://www.capacitorguide.com/coupling-and-decoupling/. [Accessed 26 May 2018].

[12] "Through-Hole vs. Surface Mount," Optimum Design Associates, 2018. [Online]. Available: http://blog.optimumdesign.com/through-hole-vs-surface-mount. [Accessed 2018 May 26].

[13] "Generic Standard on Printed Board Design IPC-2221A," May 2003. [Online]. Available: http://www.sphere.bc.ca/class/downloads/ipc_2221a-pcb%20standards.pdf. [Accessed 26 May 2018].

[14] S. Riisgaard and M. R. Blas, "SLAM for Dummies (A Tutorial Approach to Simultaneous Localization and Mapping)," 2005. [Online]. Available: https://ocw.mit.edu/courses/aeronautics-and-

astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf. [Accessed 20 May 2018].

[15] A. Wan and R. Merwe, "The Unscented Kalman Filter for Nonlinear Estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium*, Oregon, 2000.

[16] MATLAB, "Understanding Kalman Filters, Part 5: Nonlinear State Estimators," Youtube, 17 May 2017. [Online]. Available: https://youtu.be/Vefia3JMeHE. [Accessed 21 May 2018].

Appendix A

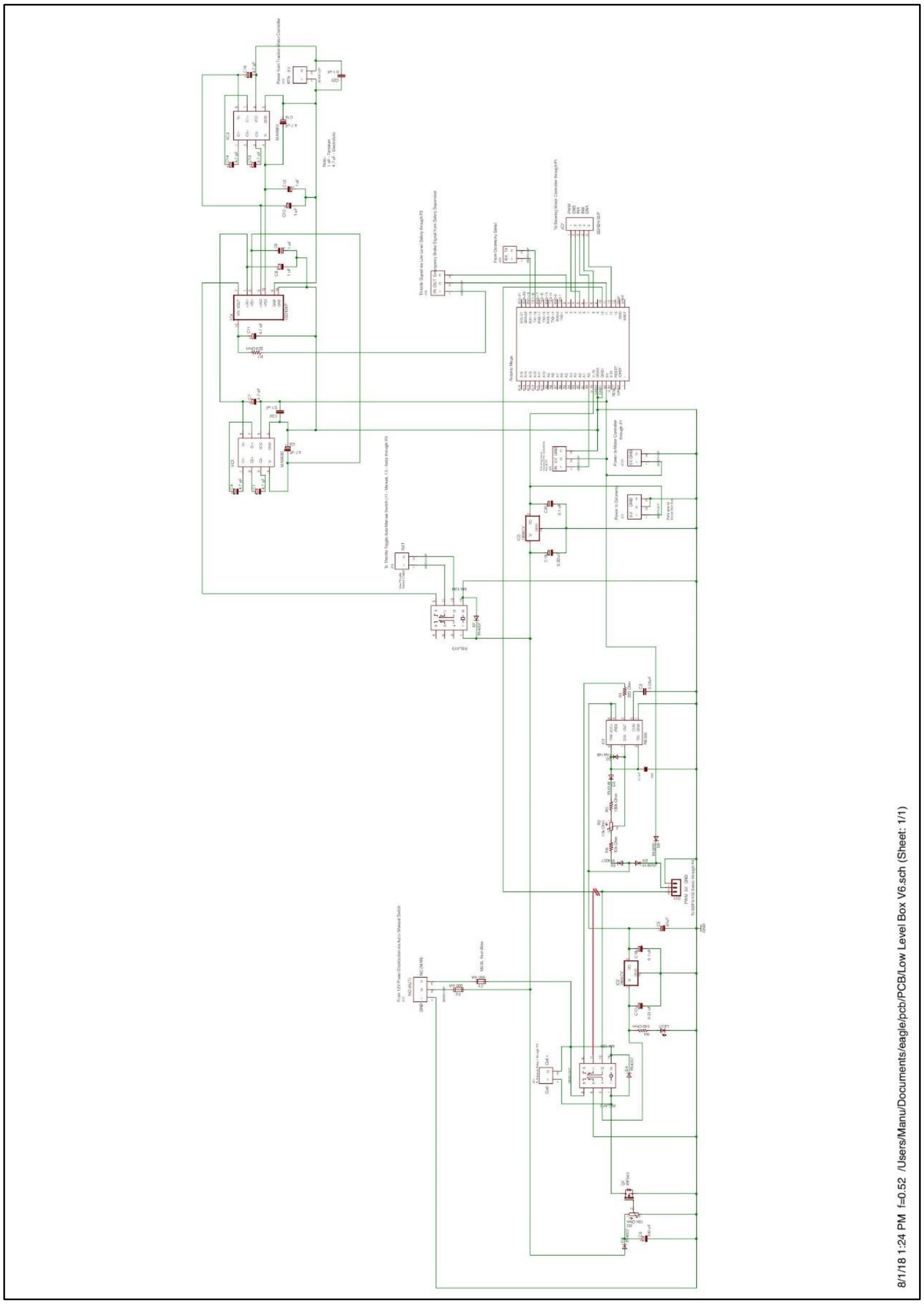
8/1/18 1:24 PM f=0.52 /Users/Manu/Documents/eagle/pcb/PCB/Low Level Box V6.sch (Sheet: 1/1)

Figure 1A, The Schematic of the Low-Level Circuit

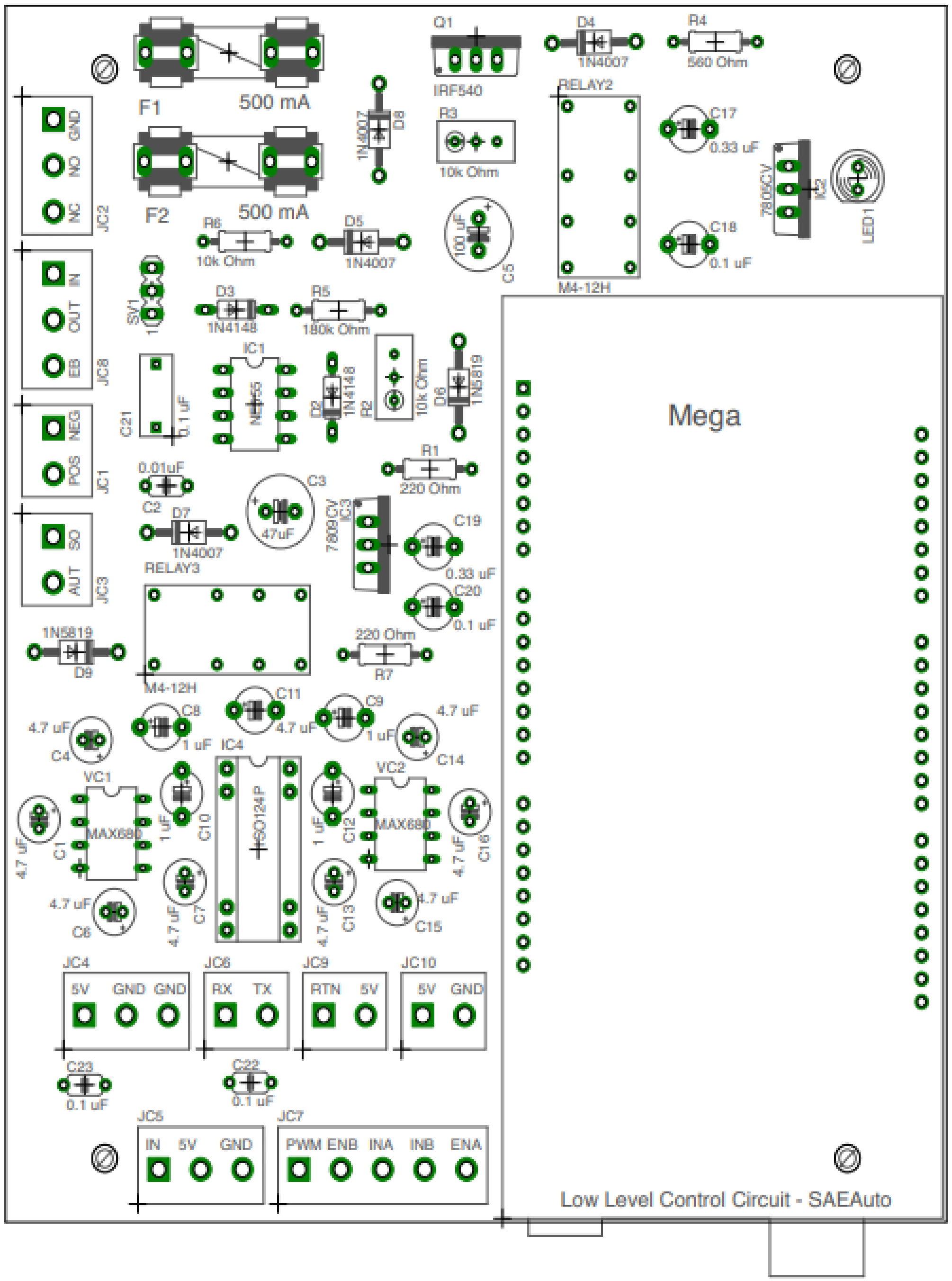

Figure 2A, The Layout of Components (Low-Level)

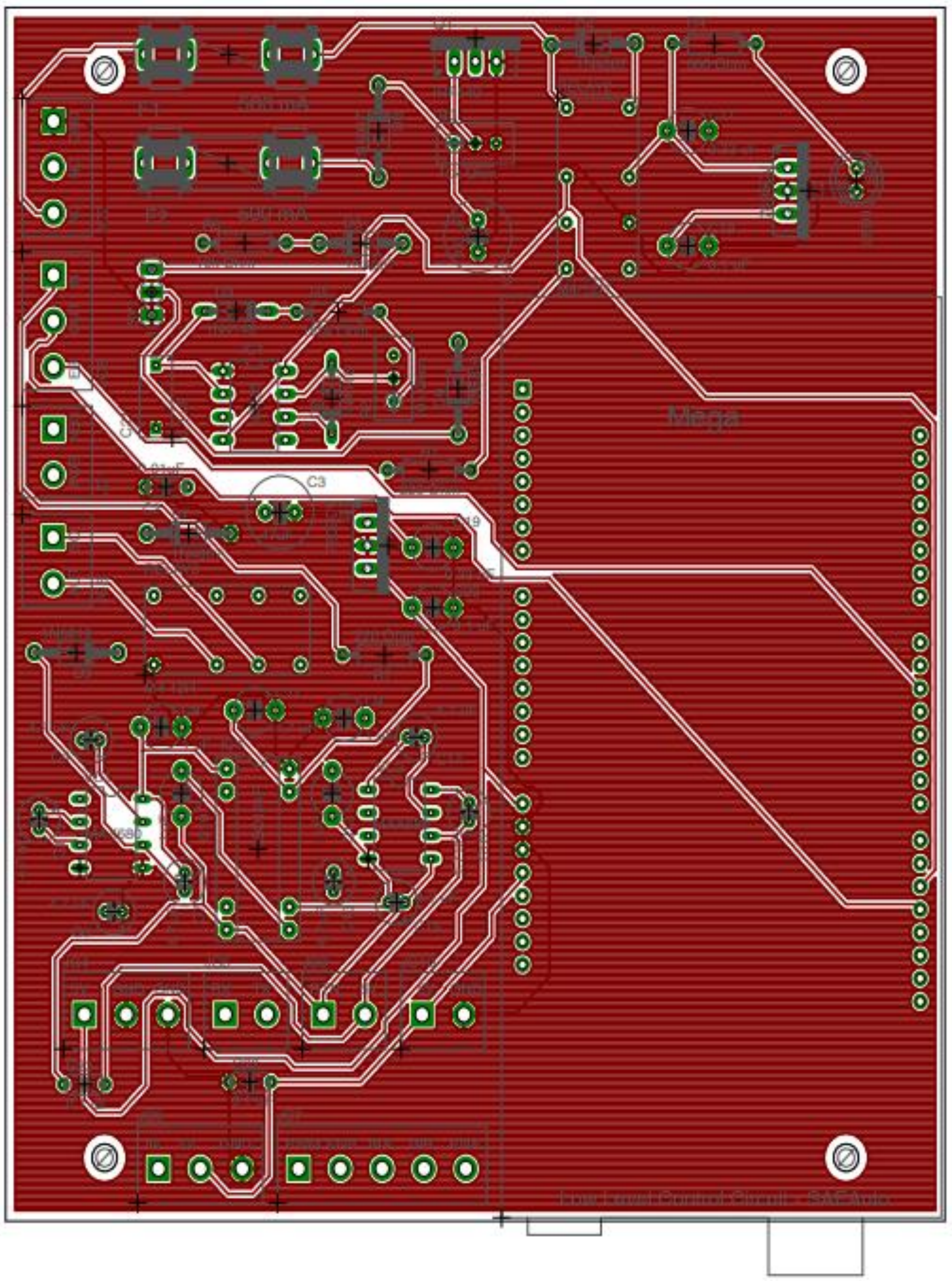

Figure 3A, Top Layer of the Low-Level PCB

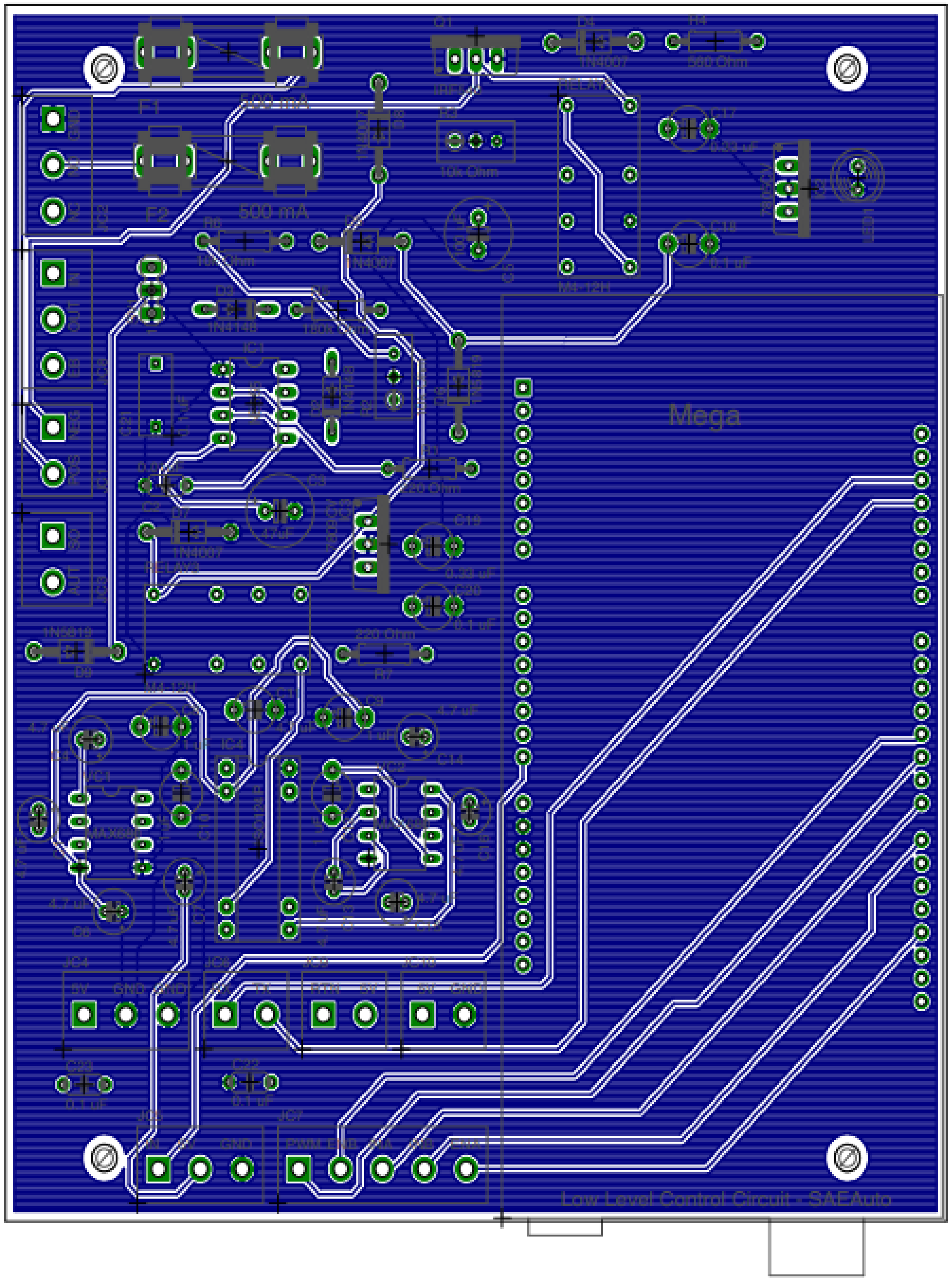

Figure 4A, Bottom Layer of the Low-Level PCB

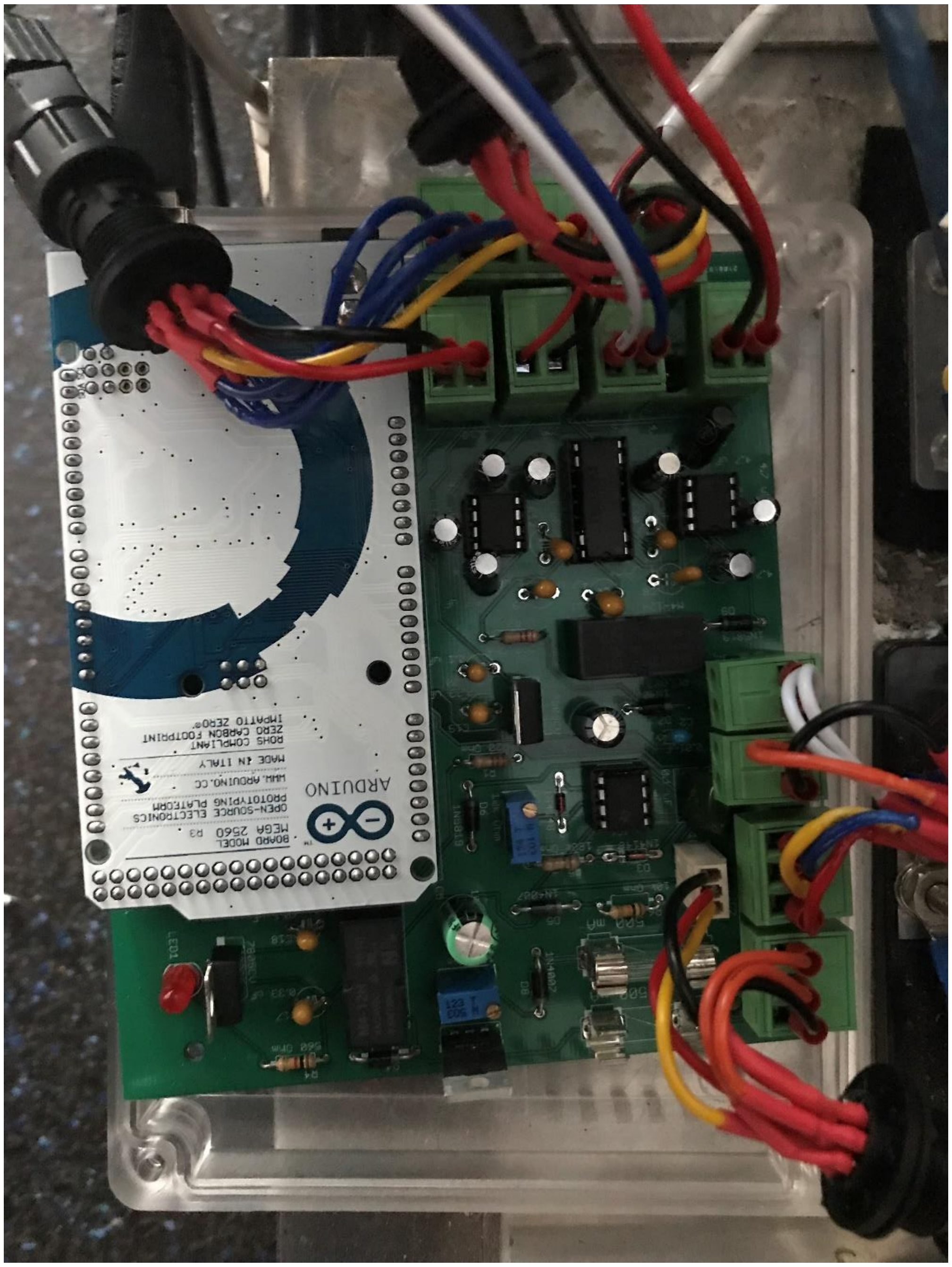
Figure 5A, The Final Constructed PCB

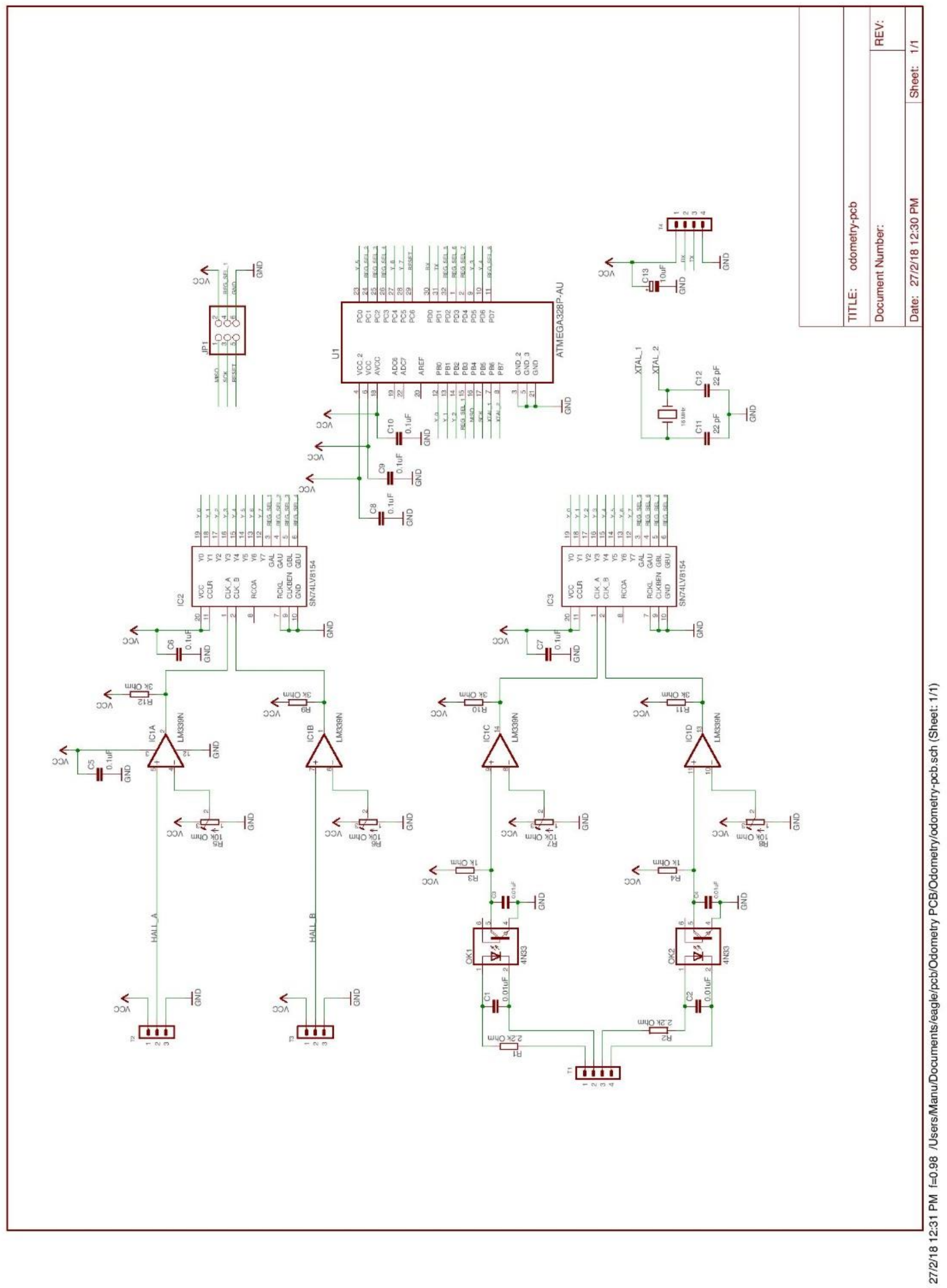

Figure 6A, Odometry Circuit Schematic

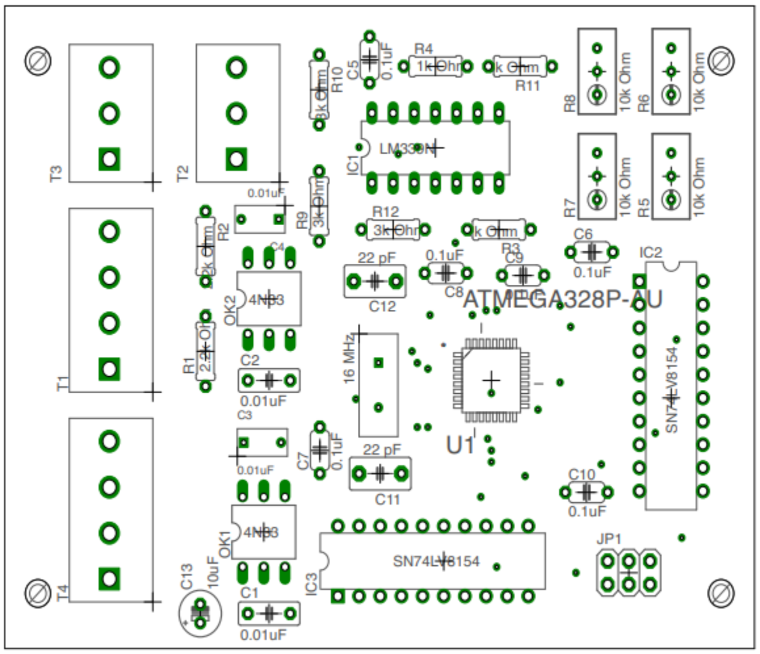

Figure 7A, Layout of Components (Odometry)

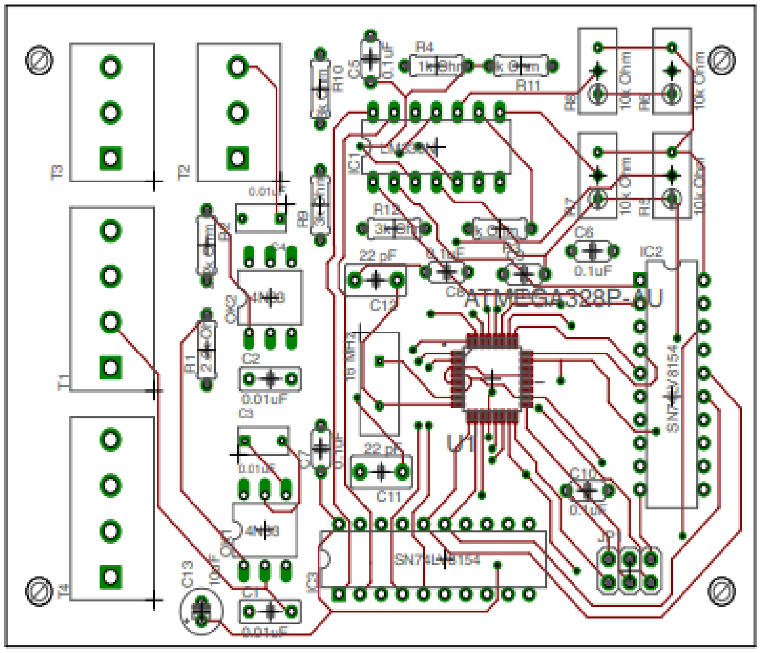

Figure 8A, Top Layer of Odometry PCB

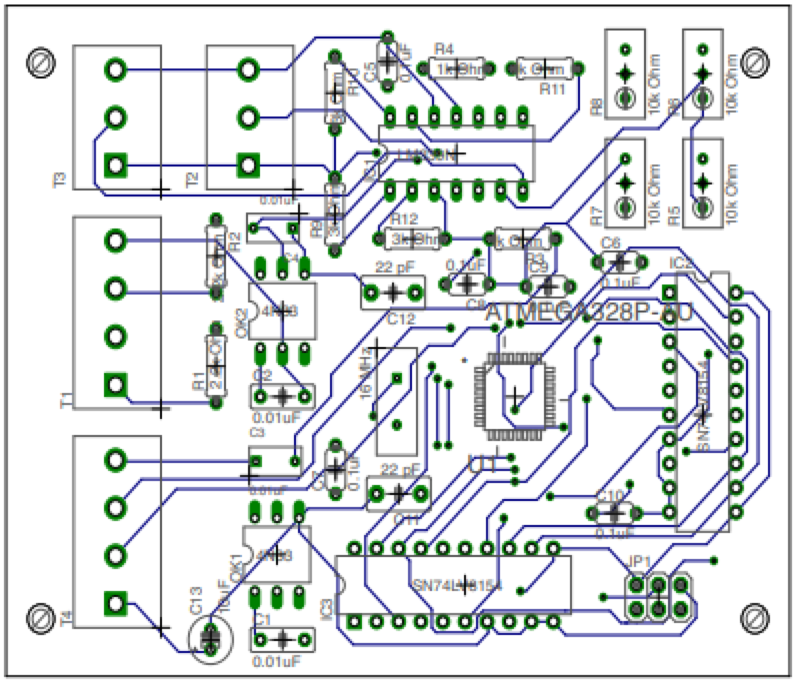

Figure 9A, Bottom Layer of Odometry PCB