LOCALIZATION FOR TEAMS OF AUTONOMOUS GROUND VEHICLES

BY

JAMES KEILLER

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Advisers:

Clinical Associate Professor William Robert Norris
Assistant Professor Katherine Driggs-Campbell

Abstract

This thesis presents the progress of work toward creating a localization engine for a fleet of autonomous ground robots using ultra-wideband (UWB) beacons in concert with the robots' on-board IMUs and encoders. The error observed from the uncorrected UWB ranges was quantified for line-of-sight conditions, and a calibration procedure was developed and applied for UWB range correction. Four localization approaches were implemented and tested on a ground robot in an outdoor environment. Dead reckoning was used as a localization baseline. An extended Kalman filter (EKF) using the manufacturer firmware's calculated positions for the measurement input was implemented and resulted in a final pose error of 29 cm. A second EKF using calibrated ranges and least squares estimation (LSE) for state measurement was implemented and resulted in a final pose error of 26 cm. A third Kalman filter, using an experimental extension of a mesh relaxation technique for position estimate, was tested but found not to track the position of the robot well. Discussion of implementation experience and recommendations for future work are provided.

Acknowledgments

I would like to thank my advisers, Professor Norris and Professor Driggs-Campbell, who shared their experience, time, talent, and resources to grant me this opportunity.

I would like to thank my peers, Ayush Rajput and Orson Wang, whose assistance in running my experiments, sometimes well past midnight, is much appreciated.

I would like to thank my mentors, Bob Striegl and John Bailey, who helped me to grow as an engineer.

I would like to thank my family, especially my grandfather, Ed Zirbes, for helping me decide to take the leap to return to school.

Finally, I offer special thanks to Serena Gountanis, Larissa Almanza, and Ayesha Ardelia. Their unwavering friendship and support mean the world to me.

Contents

List of Abbreviations vi

1 Introduction 1

2 Literature Review 4

3 Description of Research Results 6

3.1 System overview 6

3.2 Details of equipment used in experiments 7

3.2.1 Mobile robot 7

3.2.2 Sensors for inter-robot localization: Ultra-wideband (UWB) beacons 7

3.2.3 Tripods 9

3.2.4 Embedded computer for communication with UWB beacons 9

3.2.5 Power supply for the embedded computer 10

3.2.6 Main computer 10

3.3 Measurement strategies 10

3.3.1 Least squares estimation (LSE) 10

3.3.2 Intersection of spheres (Decawave firmware's strategy) 12

3.3.3 Modified spring mesh measurement 13

3.3.4 Dead reckoning 15

3.3.5 Extended Kalman filters 16

3.3.6 Measurement incorporation 18

3.4 Open-air accuracy check 21

3.5 Through-wall accuracy check 23

3.6 Spring measurement test in simulation 24

3.7 Linear calibration of DWM ranges 25

3.8 Outdoor tests 26

3.9 Operational experience 29

3.9.1 Failures of Decawave units 30

3.9.2 Behavior of a DWM tag when it loses sight of all anchors 30

3.9.3 Unexpected behavior of DWM shell 30

3.9.4 Variation in range ability 31

3.9.5 False information reported by DWMs 31
3.9.6 Problems when using portable power banks 31
3.9.7 Software conflicts 31
3.10 Code publication 32
4 Conclusion 33
References 34
Appendix A: Sample Output of DWM Shell 36
Appendix B: Sample Configuration File 45
Appendix C: DWM Range Publisher Code 46
Appendix D: Custom ROS Messages to Relay DWM Communications 61

List of Abbreviations

API	Application Programming Interface
DWM	Decawave Module
EKF	Extended Kalman Filter
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LAN	Local Area Network
LED	Light-emitting Diode
LiDAR	Light Detection and Ranging
LOS	Line-of-sight
LSE	Least Squares Estimate
NLOS	Non-line-of-sight
ROS	Robot Operating System
TDoA	Time Difference of Arrival
TLV	Type, Length, Value
ToA	Time of Arrival
UART	Universal Asynchronous Receiver / Transmitter
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
USB	Universal Serial Bus
UWB	Ultra-wideband

1 Introduction

This thesis documents the progress of work toward a localization system for a fleet of autonomous ground vehicles. Localization is the process of tracking the pose of one or more objects relative to some reference frame. When the system described herein is ready, it will allow a heterogeneous collection of autonomous agents to cooperatively localize themselves and each other, making use of ultra-wideband (UWB) beacons.

As autonomy becomes more commonplace, reliable localization systems for teams of autonomous agents become increasingly important. Indoor environments, tall buildings, trees, and more can block a GPS signal from reaching a receiver on a ground robot. This thesis proposes a localization system that uses UWB beacons as its primary sensors, and fuses UWB data with whatever additional sensors its autonomous agents are equipped. UWB beacons are relatively portable and inexpensive; the units used in the experiments described in this thesis were purchased at a cost of $300 USD for a set of 12, and are only slightly larger than a standard deck of playing cards. One potential use case is an autonomous mower equipped with wheel encoders and a camera that uses visual markers placed on its recharging station for precision location recalibration. Such a mower could take advantage of a network of nearby UWB beacons to localize itself as it moves throughout a yard and around obstacles that obstruct its line-of-sight to visual beacons, reducing pose uncertainty.

Another application would include a team of small precision agricultural robots moving between rows of crop, inspecting for pest damage and reporting problem areas with respect to a global coordinate system. Occasionally, one of the members of the team will come within range of an array of UWB beacons installed at a known location, or else see a visual marker. However, when they are outside of the visibility range of the permanently installed beacons, they can use their neighbors as landmarks to reduce the uncertainty in their location estimates, as diagrammed in Figure 1.

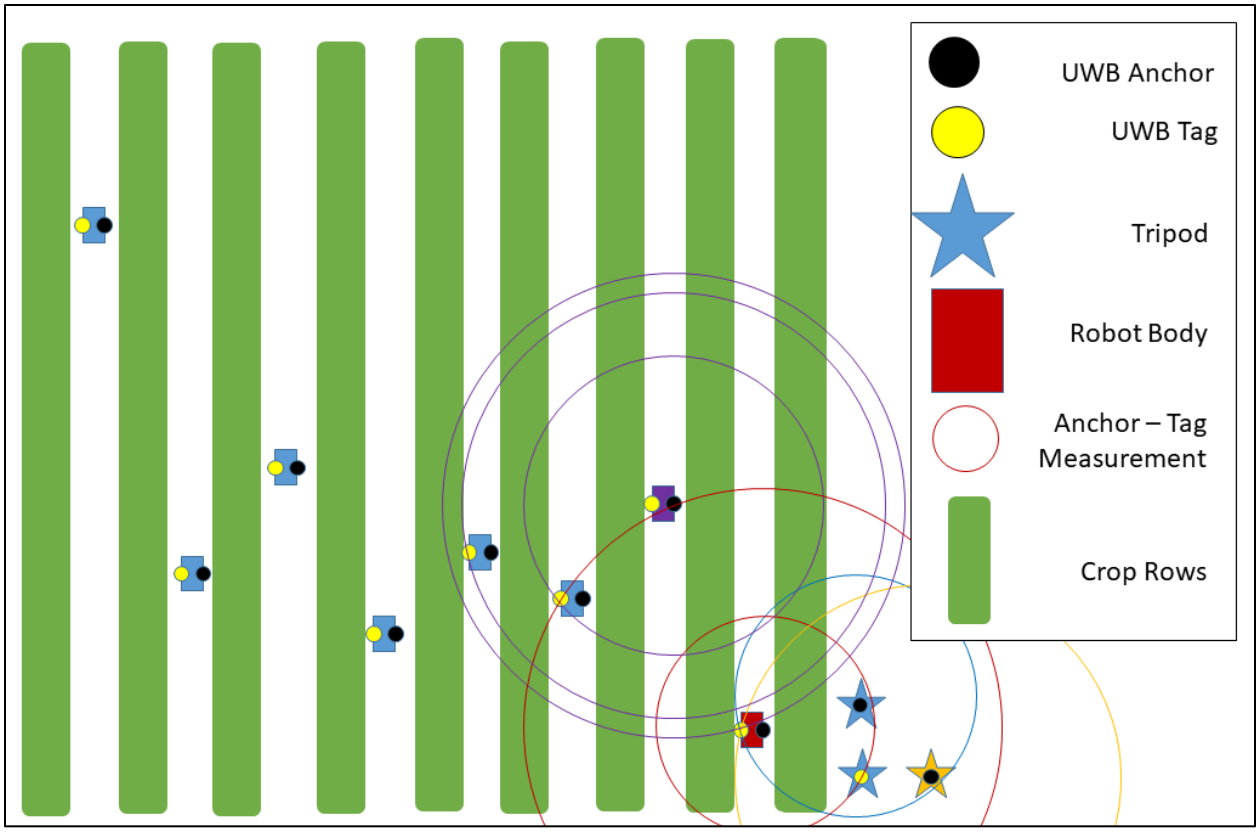


Figure 1: Autonomous agents can perform trilateration by using the beacons on their peers as landmarks when they are outside the range of permanently installed reference points. Here, small agricultural robots between rows of tall crops cannot use cameras to locate each other, but instead rely upon ultra-wideband signals.

Consider an autonomous team equipped with rotary brushes clearing snow from sidewalks. In a neighborhood with UWB beacons mounted on telephone poles at known positions, the team would have sufficient information to localize themselves to a map of neighborhood sidewalks, and spread out to cooperatively clear the paths for pedestrians. Additionally, if a robot should become stuck, its teammates could use the beacons mounted on its frame to locate it and arrange for its recovery.

Consider the benefits of running a UWB localization system in the background while operating heavy machinery while human workers wear UWB beacons on their belts. If the system identifies that a human has come too near to an intended path of travel, it could restrict the motion of heavy equipment until the condition is remedied. With a precision localization system, this identification can be done intelligently so that lockouts are triggered only by position – not simply distance, which would lead to unnecessary lockouts when humans are working in cooperation nearby. In an emergency, the same system could be used to rapidly locate all personnel that need to be evacuated from a facility.

This thesis contributes a software framework for localizing heterogeneous autonomous agents using low-cost UWB beacons. This thesis additionally contributes an extension of an earlier spring mesh pose calculation technique [1] by adding a mechanism for orientation estimation. The modified technique is evaluated for use in runtime pose estimation of a mobile ground vehicle using first simulation, and then testing with real hardware.

2 Literature Review

In Tully et al. [2], a "leap-frog" technique was used by a team of three robots. Taking turns, one robot moved while the other two remained stationary and observed the relative bearing of the mover with their cameras (four stereo camera pairs per robot). A Kalman filter combined bearing measurements with wheel encoder data from all the robots, and the system demonstrated a drift of only 1.1 m over a travel distance of 140 m.

In Buchan [3], teams of sensor-equipped "observer" robots and marker-equipped "picket" robots worked together to localize each other. They used an "inchworm" strategy that allowed multiple robots to move simultaneously, so long as there were one or more stationary robots that would act as landmarks. The observer robots were equipped with cameras and IMUs, and the picket robots were equipped with markings and LEDs that could be identified by the observers, but had no odometry themselves beyond motor feedback for speed control. Using a Kalman filter, they demonstrated localization of their robots within 0.14 m position error and 2.18° orientation error over a path length of 5 m. Unlike in the works of Tully and Buchan, the localization engine of this thesis work should avoid constraining the localized agents to taking turns moving. Additionally, the ultra-wideband beacons used in this work do not directly provide bearing information, only ranges.

In Howard et al. [1], the authors present a "mesh relaxation" technique for localization. They used SICK scanning laser range-finders as beacon detectors and place beacons with bar codes, so that the detectors could identify both the range and the bearing of beacons relative to their own local coordinate frames. When network elements sense landmarks in their environment, they report their measurements relative to their own local coordinate frames. When multiple network elements observe the same landmark, this creates global constraints between the coordinate frames. The network elements were modeled as a mesh of rigid bodies connected by their constraints. The constraints were modeled as springs with a rest position of 0 units of distance. In an ideal world with perfect measurements, all of the constraints could be satisfied simultaneously for some configuration mapping elements to the global coordinate frame. Because there are inevitably disagreements due to noise and limitations of precision, the springs will be stretched. The authors used an iterative method of "relaxing" the springs by re-aligning the coordinate frames in small steps that reduced spring tension until an equilibrium or stop condition was reached. Their approach was not implemented in real time, but instead was used on the trajectory of measurements through the entire time of data collection, as a post-processing technique. The authors identified two key limitations of their approach – the first is that the beacons needed to have unique identities, so it is unsuitable for natural landmarks which may be ambiguous to each other. The second limitation is that the size of the mesh grows linearly with time, though the authors foresaw that a modification of their strategy to remove "older" parts of

the mesh could overcome this. An important advantage of their approach is that its computation time scales linearly with N, the number of elements in the network. A Kalman filter that creates a new row in its measurement matrix for each possible landmark, on the other hand, must invert matrices of order N^2, and Markovian systems must maintain probability distributions in $3N$ dimensions.

There are some differences between the approach of Howard et al. and the spring filter approach attempted in this thesis [1]. The former used sensors capable of returning both range and bearing, whereas the ultra-wideband beacons deliver only ranges. Additionally, they made use of the data collected throughout the entire duration of an experiment to increase the accuracy of the position estimate at each moment in time. In contrast, this thesis work attempted to provide an online localization engine that would run in real time.

In Benini et al. [4], an inertial measurement unit (IMU) and ultra-wideband (UWB) beacons were used as inputs to an extended Kalman filter. By including bias in the IMU as part of the Kalman filter's state estimate, the authors report they were able to attain an error amplitude less than the resolution of their UWB sensor array (~15 cm). The UWB array used in their work took advantage of Ethernet connections to each of the UWB receivers in order to calculate both time difference of arrival (TDoA) and angle of arrival (AoA). The UWB beacons described in this thesis, in contrast, do not require synchronization. In Zhang et al. [5], non-line-of-sight (NLOS) propagation error of UWB beacons was modelled in an extended Kalman filter as a nonnegative exponential random variable. In simulation, the error of their EKF did not exceed 0.3 m under line-of-sight (LOS) conditions, and was recorded at approximately 1 m under NLOS conditions. Integrating NLOS error correction has not yet been performed with the system in this thesis, but the results documented for LOS conditions were measured with actual hardware, instead of in simulation.

3 Description of Research Results

3.1 System overview

A framework of the localization engine code was built, though the system is not yet complete. ROS, the Robot Operating System [6], is used to enable components of the system to communicate with each other. Robots in the system publish to topics in ROS, making it known what data they have available to share, such as sensor information. They can also subscribe to topics that they need information from, such as teleoperation commands. The ROS master (in experiments, the mobile robot's processor was designated as the ROS master) coordinates between members of the network, but once they are connected, they transfer information peer-to-peer.

Ultra-wideband beacons will be attached to anything that participates in localization with the system. This includes tripods, which provide reliable, unchanging position references. It includes ground vehicles, which are expected to do most of the heavy work. Ground vehicles can carry both anchor and tag Decawave modules (DWMs), so that they can create a chain of localized reference points from the vehicles closest to the tripods, out toward the vehicles farthest away. Unmanned aerial vehicles (UAVs) can also carry localization beacons to improve the accuracy of the network as they perform their other duties, as the beacons are lightweight, weighing approximately 100 grams. As the fleet spreads out, robots with high confidence in their position may place additional beacons on the ground and report their position to the localization system. This action will provide reference beacons with high confidence and allow the fleet to travel great distances from the tripods that define the reference frame. Finally, beacons may be placed on materials that are being moved by the fleet that need to be tracked. For example, if a long bridging platform is laid across an obstacle such as a ditch, agents on each side of the obstacle will be able to find the ends of the platform with precision by localizing their beacons.

The system is designed with modularity in mind. The system defines a Python class of a `Body`, and all implementations of bodies must handle function calls appropriate for the localization filters being used, such as reporting beacon position, or updating pose estimates. The three body architectures already implemented are `Tripod`, `FreeBody`, and `Jackal` ("Jackal" is the name of the ground robot used in the experiments). Each architecture is a subclass of the `Body` class. Rigidly mounted beacons at reliable, static positions are represented by the `Tripod` subclass. The `Jackal` class is a prototype for mobile ground robots that fuse UWB measurements with IMU and encoder measurements. The `FreeBody` class can be used to represent humans with UWB beacons attached to their belts. Tracking human personnel can help autonomous agents plan their movements to avoid conflicts with humans, and allow personnel to be located quickly in the event of an emergency. More architectures will be added in the future to represent other vehicles. When a user prepares to use the localization system, they can specify the identities and architectures of all agents in the fleet with a configuration file. That same file specifies which

DWM beacons are mounted on which bodies, and can provide calibration information for the beacons that are being used. Beacon-to-body assignments are stored in a Python dictionary, so when filters need to reference them they can be found with a hash lookup instead of stepping through a list. See Appendix B for an example of a configuration file.

3.2 Details of equipment used in experiments

3.2.1 Mobile robot

The robot used in the experiments described in this document was a Jackal Unmanned Ground Vehicle (UGV) from Clearpath Robotics [7]. The Jackal is a four-wheeled, electric differential-drive ground robot 508 mm long, 430 mm wide, 250 mm tall, measuring 17 kg. The Jackal is equipped with its own wireless router, and has an onboard computer that communicates sensor readings through ROS messages. The sensors on the Jackal used in experiments were the IMU and the wheel encoders, as these are common proprioceptive sensors, representative of what commercial mobile robots should have, as a minimum. The LiDAR and GPS capabilities of the Jackal were not used at any time during the experiments performed in this thesis work, as the intent was to provide a localization framework that would not be dependent upon either of them.

Modularity was important when designing the localization engine, as it needs to scale to arbitrary robot types in the future. A high level of modularity was achieved by refraining from any modifications to code or other files stored on the robot itself during the implementation of this system. To meet this constraint, an embedded computer was mounted to the top of the Jackal to interface with the DWM modules that were attached to its port and starboard side. The computer communicated with the DWMs and published their sensor data to the Jackal's ROS network.

3.2.2 Sensors for inter-robot localization: Ultra-wideband (UWB) beacons

Proprioceptive sensors measure values internal to the system [8], such as wheel rotation, joint angles, battery charge, etc. Exteroceptive sensors measure values related to the robot's environment, such as measurements of distances and bearings to landmarks, camera images, LiDAR scans, etc. Both proprioceptive and exteroceptive sensors are important tools in localization, but in order for a network of robots to work together most effectively, the framework should make extensive use of exteroceptive sensors, so that a misguided robot can always be corrected by its peers.

To coordinate the fleet of autonomous agents effectively, the master controller node must be able to obtain location information of the agents to a reasonable level of granularity. Ultra-wideband (UWB) beacons are a good choice for the primary sensors to fulfill this need. The benefits of UWB beacons are discussed in the next section. There are many sensors that are valuable to robots fulfilling a specific purpose, but would not be suitable as the basis for an inter-robot localization network. One of the key strengths of two-way wireless communication is the ability

to resolve landmark ambiguity with a definitiveness that other sensing methods struggle to match. Depending on their role, robots in the autonomous fleet will have varying levels of sensing capabilities. Cameras were used in the literature for inter-robot positioning, but the performance of camera-based recognition can be impacted by mud, dust, dirt and water on lenses, smoke, fog, and changing light conditions [3]. Even excellent image recognition neural networks can provide false positives or false negatives under the right conditions [9]. With properly encrypted wireless communications, the identity of the object being tracked is known definitively, even if there may be some error in its position estimation. LiDAR data would also be difficult to implement as the basis for a localization system. A robot coming across another member of the fleet would have difficulty discerning the identity of its neighbor through LiDAR alone if there were more than one identical to it in shape and size in the fleet.

The UWB beacons used in this document are Decawave MDEK1001 units which come equipped with DWM1000 modules along with interfacing hardware and firmware. The Decawave modules (DWMs) use open-source firmware, and can communicate through a serial port connection, a UART type-length-value (TLV) interface, a UART shell environment, or through Bluetooth to an Android application provided by the manufacturer. The DWMs are relatively lightweight, weighing approximately 100 g apiece without battery, and multiple units can be mounted on the Jackal without any noticeable impact to its mobility, as shown in Figure 2.



Figure 2: Jackal ground robot with DWMs mounted on left and right side.

3.2.2.1 Ultra-wideband communication

Ultra-wideband communication devices use extremely narrow pulses to communicate between transmitters and receivers. As discussed in Nekoogar [10], these short-duration pulses generate a very wide bandwidth, and with modern microprocessing and fast-switching semiconductors, UWB communication can operate at very low average power levels, often below the noise floor of other transmission systems. Because of this property, UWB offers advantages over other forms of

wireless communication, including resistance to signal loss caused by multipath transmission, robustness to jamming, and coexistence without interference of other radio services.

3.2.2.2 Symmetric double-sided two-way ranging

The DWMs use two-way ranging to determine the distance between beacons. Each beacon is configured as either an anchor or a tag. Unpaired tags send polling messages, and if an anchor receives the polling message and chooses to range with the tag, it will send a ranging initialization message. The tag then responds to that message with a message that contains the time of delay from when it received the first message to when it sent its own message. The anchor, upon receiving this message, then can calculate the propagation time for an RF signal to propagate between the two beacons twice by subtracting the delay time from the total time observed between its ranging initialization message and receipt of the tag's response.

$$\text{range} = \frac{t_{\text{receive response}} - t_{\text{sent initialization}} - \tau_{\text{tag's delay}}}{c}, \qquad c = \text{speed of light} \tag{3.1}$$

The anchor, similarly, sends a message to the tag, which contains information on the delay between the anchor's receipt of the tag's message and the time it sent the final message. When the tag receives this message, it can also calculate the range between it and the anchor. Because each device self-reports its own processing delay before responding, each beacon can calculate the total transmission time by itself, without need of synchronized clocks.

3.2.2.3 Type, length, value (TLV) communication

The DWMs provide UART access to a TLV interface through their micro-USB connectors. TLV stands for Type, Length, Value. Messages are sent and received in binary format, with the first two bytes of each message sent specifying the Type of the message, as specified in the device manufacturer's API guide [11]. The second two bytes specify the Length of the value part of the message. The third part of the message, the Value, is Length bytes long.

3.2.3 Tripods

To place DWM units in repeatable, precise positions in outdoor environments, TYCKA 47-inch tripods with spring-armed phone holders were used. The DWM unit could be mounted in the phone holder, and the tripod adjusted until the bubble level indicated the DWM was pointing exactly perpendicular to the ground. The tripods also have a hook underneath, and during setup a weight on a string was suspended from the hooks to ensure the tripods were positioned directly over their marked locations on the ground.

3.2.4 Embedded computer for communication with UWB beacons

A Raspberry Pi 3b+ was chosen as the embedded computer. It has Ubuntu MATE 18.04 and ROS Melodic installed. The Raspberry Pi was mounted on the top center of the Jackal, and was

connected through USB UART to the DWMs on the port and starboard sides. After using its built-in Wi-Fi adapter to connect to the Jackal's router and join the ROS network hosted by the Jackal, the Raspberry Pi starts two instances of the `dwm_publisher` ROS node written for this project. Those nodes repeatedly ask the attached DWMs to call the TLV command `dwm_loc_get` (0x0C), and publishes the response in the form of a custom ROS message type to the ROS network. See Appendix C for the implementation of the publisher code.

3.2.5 Power supply for the embedded computer

A Beston 8000 mAh cellphone portable charger was selected to provide the power to the Raspberry Pi 3b+ and the two DWMs it communicated with. They are advertised to provide 5 V at 2.1 A through each of two USB ports.

3.2.6 Main computer

The main computer used for filter computation was a laptop equipped with a 4th Generation Intel i7-4700MQ Processor (2.4 GHz) running Ubuntu 18.04 Bionic Beaver, with 24GB DDR3 SDRAM at 1600 MHz. Its wireless card for communicating with the Jackal's network was a Bigfoot Networks Killer™ Wireless-N 1103 - 802.11A/B/G/N Wireless LAN Module.

3.3 Measurement strategies

3.3.1 Least squares estimation (LSE)

When solving for the position of a DWM tag receiving ranges from anchors, the least square estimation technique described in Blewitt [12] can be imitated. In this application, there is no need to estimate τ, the clock bias of the tag, because two-way ranging is used. This allows a 3D position calculation to be performed with as few as three ranges, instead of the four ranges GPS requires. To start, assume an observed range can be written as the sum of the modelled range, plus an error term, as shown in Equation (3.2):

$$r_{observed} = r_{model} + noise = r(x, y, z) + v \tag{3.2}$$

Applying Taylor's theorem, expand about the model computed using provisional parameter values (x_0, y_0, z_0) and discard second and higher order terms.

$$r(x, y, z) \cong r(x_0, y_0 z_0) + (x - x_0)\frac{\partial r}{\partial x} + (y - y_0)\frac{\partial r}{\partial y} + (z - z_0)\frac{\partial r}{\partial z} \tag{3.3}$$

$$= r_{computed} + \frac{\partial r}{\partial x}\Delta x + \frac{\partial r}{\partial y}\Delta y + \frac{\partial r}{\partial z}\Delta z \tag{3.4}$$

The residual observation is the difference between the actual observation and the modelled observation computed using the provisional parameter.

$$\Delta r = r_{\text{observed}} - r_{\text{computed}} = + \frac{\partial r}{\partial x}\Delta x + \frac{\partial r}{\partial y}\Delta y + \frac{\partial r}{\partial z}\Delta z + v \tag{3.5}$$

One such equation is obtained for each of the i anchor ranges received by the tag. A matrix is constructed from the system of equations as shown in (3.6).

$$\begin{bmatrix} \Delta r^1 \\ \cdots \\ \Delta r^i \end{bmatrix} = \begin{bmatrix} \frac{\partial r^1}{\partial x} & \frac{\partial r^1}{\partial y} & \frac{\partial r^1}{\partial z} \\ \cdots & \cdots & \cdots \\ \frac{\partial r^i}{\partial x} & \frac{\partial r^i}{\partial y} & \frac{\partial r^i}{\partial z} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} + \begin{bmatrix} v^1 \\ \cdots \\ v^i \end{bmatrix} \tag{3.6}$$

Note that, using the default Decawave firmware, the number of ranges i will either be 3 or 4, as 2 is too few, and no more than 4 ranges are returned by the firmware. However, the number of ranges could potentially be increased by modifying the firmware, or by implementing a purpose-built circuit board incorporating the DM1000 chip.

Continuing to follow the guidance in Blewitt, Equation (3.6) can be re-written as (3.7):

$$b = A\vec{x} + \vec{v} \tag{3.7}$$

to express the linear relationship between the residual observations b and the unknown corrections to the parameter $\vec{x}$. The column matrix $\vec{v}$ contains the noise terms. The matrix A is known as the "design matrix" and its coefficients can be computed as shown in (3.8).

$$A = \begin{bmatrix} \frac{x_0 - x^1}{r_1} & \frac{y_0 - y^1}{r_1} & \frac{z_0 - z^1}{r_1} \\ \cdots & \cdots & \cdots \\ \frac{x_0 - x^i}{r_i} & \frac{y_0 - y^i}{r_i} & \frac{z_0 - z^i}{r_i} \end{bmatrix} \tag{3.8}$$

Let $\hat{x}$ be a solution for the linearized observation equations. The estimated residuals are the difference between the actual observations and the model observations given the estimate.

$$\hat{v} = b - A\hat{x} \tag{3.9}$$

The least squares solution refers to the value of $\hat{x}$ that minimizes the function shown in Equation (3.10).

$$J(x) = \sum_{i=1}^{m} v_i^2 = v^T v = (b - Ax)^T (b - Ax) \tag{3.10}$$

The solution, as derived in Blewitt [12], can be found as

$$\hat{x} = (A^T A)^{-1} A^T b \tag{3.11}$$

The computed $\hat{x}$ is considered to be the measurement of the tag's position, when this method is selected in the localization engine.

3.3.2 Intersection of spheres (Decawave firmware's strategy)

Given a measurement from a DWM, the range reported between a tag and an anchor at a known location represents a sphere of possible locations around that anchor. In the case of physical beacons, it is a reasonable assumption that the spheres will not be concentric, as two solid beacons cannot occupy the same space.

The intersection of two non-concentric hollow spheres must be one of three cases: an empty set (the spheres do not intersect), a single point (the spheres intersect in exactly one location), or a circle (the spheres overlap). The intersection of three non-concentric hollow spheres must then be an empty set, the intersection of a point with a circle, the intersection of two points, or the intersection of two circles. The intersection of a point with anything else is either that point, or an empty set. The intersection of two non-concentric hollow circles is either an empty set, a single point or two points.

The firmware source code provided by Decawave, in the file "trilateration.cpp" , appears to have been set up in preparation for calculating the position of a point using the intersection of three spheres method. If the intersection of the three spheres is two points, then a line between the two

points is calculated, and the intersection of the fourth sphere with that line is meant to be the point in space returned by the trilateration code. If the first three spheres do not intersect, then the code incrementally increases the radius of the calculated spheres to seek an intersection. However the fourth range does not appear to be used. In the function `GetLocation()`, the result is calculated with a function call to `deca_3dlocate`, but the radius of the first sphere is passed in the argument where that represents the radius of the fourth sphere. See Table 1.

Table 1: Referenced Firmware Source Code

Function declaration	`int deca_3dlocate (vec3d *const solution1,`
	`vec3d *const solution2,`
	`vec3d *const best_solution,`
	`int *const nosolution_count,`
	`double *const best_3derror,`
	`double *const best_gdoprate,`
	`vec3d p1, double r1,`
	`vec3d p2, double r2,`
	`vec3d p3, double r3,`
	`vec3d p4, double r4,`
	`int *combination)`

Function call using r1 instead of r4	`result = deca_3dlocate (&o1, &o2, best_solution, &error, &best_3derror, &best_gdoprate, p1, r1, p2, r2, p3, r3, p4, r1, &combination);`

3.3.3 Modified spring mesh measurement

The relaxation-on-a-mesh technique described in Howard et al. [1] tackled a generalized localization problem, and an attempt was made to expand upon that technique. Let z'_a be the pose of some object in the local coordinate system of a, and let z'_b be the pose of that same object in the local coordinate system of b. In some arbitrary global coordinate system, those two points must map to the same point. Let Γ be a coordinate transformation operator that maps from a local to a global coordinate system. Equation (3.12) gives the constraint:

$$\Gamma_a z'_a - \Gamma_b z'_b = 0 \tag{3.12}$$

Further, consider a pair of rigid bodies a and b that are joined by a spring. Let x_a be the pose of body a and x_b be the pose of body b in the same global coordinate system. The spring joins

the point z_a' on body a to the point $z_b{}'$ on body b. The energy U_{ab} of the spring is given by Equation (3.13).

$$U_{ab} = \frac{1}{2} k_{ab} \left(\Gamma_{x_a} z_a' - \Gamma_{x_b} z_b' \right)^2 \tag{3.13}$$

Howard et al. considered a network of sensors as a mesh of many such rigid bodies, with many objects that are mapped to various local coordinate frames. They set the spring constant k_{ab} to $1/\sigma^2$, where σ is the uncertainty in the measurement the spring represents. The total energy of such a system is given by Equation (3.14).

$$U = \frac{1}{2} \sum_{ab} U_{ab} \tag{3.14}$$

The idea is to find the set of poses that minimizes the energy of the system. A real system of springs and masses would perform this relaxation automatically, settling into an equilibrium state where all net forces on all bodies are zero. To simulate the process, the authors derived an update rule: For each mass a, update the pose x_a using the rule:

$$F_a = \sum_b F_{ab} = \text{total force acting on } a \tag{3.15}$$

$$x_a \leftarrow x_a + \frac{1}{2} \nabla t^2 F_a \tag{3.16}$$

The algorithm iteratively updates the force calculations based on positions, and updates positions based on forces, until convergence criteria are met.

Note that ∇t^2 is a constant, and ∇t is an arbitrary constant chosen for a time step, and may be considered as a tuning parameter. The "mass" assigned to each rigid body is an opportunity for tuning as well. Note also that Howard et al. did not mention calculations of torque on the rigid bodies, just displacement. In this thesis work, a modified version of the algorithm was implemented, that allows each body to respond to forces in a way that is appropriate to the architecture of the agent represented. Tripods were assumed to be at known positions, and so while their observations can generate forces on other bodies, their position is not changed during the update step. A class called `FreeBody` was implemented in the code, which has no process

update step, and has a `can_push` attribute set to `False`. This attribute setting means range measurements from instances of this class do not exert forces on other bodies, but those instances can be pushed around freely based on measurements by the filter. A body class representing the Jackal was implemented. In addition to calculating the net force on the Jackal, torque is calculated as if the force were applied directly to the beacons on the left and right sides of the Jackal. The rotational inertia of the Jackal is computed as if it were a rectangular prism of the same dimensions of the Jackal, on a frictionless plane.

$$\text{Jackal Rotational Inertia} = \text{mass} * \frac{(\text{height}^2 + \text{length}^2)}{12} \tag{3.17}$$

A body architecture representing UAVs has not yet been implemented in the code, but is planned for the future.

The intent of the localization system using the spring measurement is to equip vehicles with DWMs configured as both anchors and tags. In this way, a small number of stationary tripods can provide reliable position beacons to ground the global coordinate system, but the fleet can chain its measurements together from vehicle to vehicle as they spread out, and reach distances well outside of the broadcast range of the DWMs mounted on the tripods. Such a system could work above or below ground equally well, as long as sufficient agents were present to chain the measurements together. In this document, the measurement calculation was fed into a Kalman filter, but in Howard et al. [1] the authors point out that the computation time of the mesh algorithm scales linearly, and so does not become intractable with a large number of agents. Once the modified spring mesh code has been fully implemented, it can be used to compute pose measurements globally in linear time. Each agent will then input the globally calculated poses of its attached beacons as a state measurement into its EKF. Each agent's EKF models only the agent's pose, not the pose of its neighbors. This decentralization will allow for tractable computation, while appropriately weighing the measurement updates against the process updates using a Kalman gain.

3.3.4 Dead reckoning

Dead reckoning is the process of accumulating measurements of relative position changes over time, from some initial starting pose estimate. As there is no direct measurement of state – only change in state – error is expected to accumulate over time, unless additional methods are combined with dead reckoning to estimate the state and correct the error. For the Jackal, the state vector is as follows:

$$\vec{x} = \begin{bmatrix} x \\ y \\ z \\ \theta \end{bmatrix} = \begin{bmatrix} displacement\ of\ Jackal\ along\ x\ axis \\ displacement\ of\ Jackal\ along\ y\ axis \\ displacement\ of\ Jackal\ along\ z\ axis \\ counterclockwise\ rotation\ of\ Jackal\ about\ the\ z\ axis \end{bmatrix} \tag{3.18}$$

Let ω be the angular velocity of the Jackal as reported by the IMU. The process variables are shown in Equation (3.19).

$$\vec{u} = \begin{bmatrix} u \\ \omega \end{bmatrix}, \qquad u = \frac{\text{left motor speed} + \text{right motor speed}}{2} \tag{3.19}$$

The differences between the idealized process model and reality is accounted for by adding Gaussian noise. Let $w(t)$ be a zero-mean Gaussian random vector with covariance matrix G. The motion of the Jackal is modelled in Equations (3.20) and (3.21):

$$\dot{\vec{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} u * \cos(\theta) \\ u * \sin(\theta) \\ 0 \\ \omega \end{bmatrix} + w(t) \tag{3.20}$$

$$\vec{x}_{new} = \vec{x}_{old} + \Delta t * \dot{\vec{x}} \tag{3.21}$$

Although the Jackal has four wheels, it publishes its motors' speed measurements to its ROS network as a single left speed and a single right speed under the `/feedback` topic. Using ROS, the publishing rate of the encoder speed values was measured at approximately 19.8 Hz. The Jackal publishes its rotational velocity under the `/imu/data` topic at a rate of 76 Hz. To calculate Δt, the time of each process measurement is considered to be the time when both a linear speed and a rotational speed value have been received from the ROS network, and Δt is set equal to the time between the current process measurement and the previous one, with measured speeds treated as if constant.

3.3.5 Extended Kalman filters

The extended Kalman filter is an extension of the Kalman filter for nonlinear processes [13]. Let $\vec{x}$ represent the state vector with state estimate covariance P. Let $\vec{u}$ represent the deterministic control / process measurement, let w be process noise with covariance matrix Q, and let t be time. The rate of change of the state vector is as in Equation (3.22).

$$\dot{\vec{x}}(t) = f(\vec{x}, \vec{u}, w, t) \tag{3.22}$$

A measurement that provides information about the state is assumed to be linear as in Equation (3.23).

$$\vec{y}(t) = C\vec{x} + v \tag{3.23}$$

where v is white noise with covariance matrix R.

In an extended Kalman filter, during the process update step, the state estimate covariance matrix has the update rule shown in Equation (3.24).

$$\dot{P} = \Delta(t)P(t) + P(t)\Delta(t)^T + GQG^T \tag{3.24}$$

In Equation (3.24), $\Delta(t)$ is not a change in time, but is shorthand notation for the Jacobian of the nonlinear function $f(\vec{x}, \vec{u}, w, t)$ around the current state $\hat{x}(t)$. $\Delta(t)$ and G are given by Equations (3.25) and (3.26):

$$\Delta(t) = \left.\frac{\partial f(\vec{x}, \vec{u}, w, t)}{\partial \vec{x}}\right|_{\vec{x}(t)=\hat{x}(t)} \tag{3.25}$$

$$\mathrm{G} = \left.\frac{\partial f(\vec{x}, \vec{u}, w, t)}{\partial w}\right|_{w(t)=\vec{0}} \tag{3.26}$$

In the case of a Jackal, these partial derivatives are as shown in Equations (3.27) and (3.28).

$$\Delta(t) = \frac{\partial}{\partial \vec{x}}\begin{bmatrix} u * \cos(\theta) \\ u * \sin(\theta) \\ 0 \\ \omega \end{bmatrix} + \frac{\partial}{\partial \vec{x}} w(t) = \begin{bmatrix} 0 & 0 & 0 & -u * \sin(\theta) \\ 0 & 0 & 0 & u * \cos(\theta) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.27}$$

$$G = \frac{\partial}{\partial w}\begin{bmatrix} u * \cos(\theta) \\ u * \sin(\theta) \\ 0 \\ \omega \end{bmatrix}\Bigg|_{w(t)=\vec{0}} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.28}$$

In an EKF, when a measurement is received, the update rules for the state and state covariance are as shown in Equations (3.29) and (3.30).

$$\vec{x}_{new} = \vec{x}_{old} + K(y - C\vec{x}_{old}) \tag{3.29}$$

$$P_{new} = (I - KC)P_{old}(I - KC)^T + KRK^T \tag{3.30}$$

K is the Kalman gain, and is computed as in Equation (3.31):

$$K = P_{old}C^T(CP_{old}C^T + R)^{-1} \tag{3.31}$$

3.3.6 Measurement incorporation

If the extended Kalman filter receives state measurements in the form of pairs of position measurements for the beacons mounted on its left and right side, then the measurement $\vec{y}$ is calculated as in Equation (3.32):

$$\vec{y} = \begin{bmatrix} \frac{(x_{left} + x_{right})}{2} \\ \frac{(y_{left} + y_{right})}{2} \\ \frac{z_{left} + z_{right}}{2} \\ arctan2\left((x_{right} - x_{left}), (y_{left} - y_{right})\right) \end{bmatrix} = \begin{bmatrix} x_{meas} \\ y_{meas} \\ z_{meas} \\ \theta_{meas} \end{bmatrix} \tag{3.32}$$

In this case, the C matrix for the Kalman filter is simply the identity matrix shown in Equation (3.33).

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.33}$$

A second option for using DWM data for the measurement update is to list the i range values received by the Jackal's left DWM and the j range values received by the Jackal's right DWM, and treat the list as a single measurement. The relationship between range and x, y, z, θ is nonlinear, but Choset et al. [14] showed that a range and bearing measurement taken from the center of a robot's coordinate frame can have its relationship to position linearized well enough to construct the C matrix about a point for a Kalman filter to work. Building from that, though this implementation has no bearing measurement, a linearized C matrix can be derived for off-center range measurements, relating the ranges observed from the left and right DWM to the Jackal's pose.

Let b be the width of a Jackal robot and z be the height of a DWM module mounted on a Jackal. Then the pose of the DWM module mounted on the left side of a Jackal is as in Equation (3.34).

$$\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = \begin{bmatrix} x - \frac{b}{2}\sin(\theta) \\ y + \frac{b}{2}\cos(\theta) \\ z \end{bmatrix} \tag{3.34}$$

The range from a beacon A at location $\begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix}$ to a DWM mounted on the left side of the Jackal is shown in Equation (3.35).

$$r_A = \sqrt{\left(x_a - \left(x - \frac{b}{2}\sin(\theta)\right)\right)^2 + \left(y_A - \left(y + \frac{b}{2}\cos(\theta)\right)\right)^2 + (z_A - z)^2} \tag{3.35}$$

And so the partial derivatives can be computed as in Equations (3.36) through (3.39):

$$\frac{\partial r_A}{\partial x} = \frac{x - \frac{b}{2}\sin(\theta) - x_A}{\sqrt{\left(x_a - \left(x - \frac{b}{2}\sin(\theta)\right)\right)^2 + \left(y_A - \left(y + \frac{b}{2}\cos(\theta)\right)\right)^2 + (z_A - z)^2}} \tag{3.36}$$

$$\frac{\partial r_A}{\partial y} = \frac{y + \frac{b}{2}\cos(\theta) - y_A}{\sqrt{\left(x_a - \left(x - \frac{b}{2}\sin(\theta)\right)\right)^2 + \left(y_A - \left(y + \frac{b}{2}\cos(\theta)\right)\right)^2 + (z_A - z)^2}} \tag{3.37}$$

$$\frac{\partial r_A}{\partial z} = \frac{z - z_A}{\sqrt{\left(x_a - \left(x - \frac{b}{2}\sin(\theta)\right)\right)^2 + \left(y_A - \left(y + \frac{b}{2}\cos(\theta)\right)\right)^2 + (z_A - z)^2}} \tag{3.38}$$

$$\frac{\partial r_A}{\partial \theta} = \frac{\left(x_A - x + \frac{b}{2}\sin(\theta)\right)\left(\frac{b}{2}\cos(\theta)\right) + \left(y_A - y - \frac{b}{2}\cos(\theta)\right)\left(\frac{b}{2}\sin(\theta)\right) + (z - z_A)}{\sqrt{\left(x_a - \left(x - \frac{b}{2}\sin(\theta)\right)\right)^2 + \left(y_A - \left(y + \frac{b}{2}\cos(\theta)\right)\right)^2 + (z_A - z)^2}} \tag{3.39}$$

The pose of the DWM module mounted on the right side of a Jackal is

$$\begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = \begin{bmatrix} x + \frac{b}{2}\sin(\theta) \\ y - \frac{b}{2}\cos(\theta) \\ z \end{bmatrix} \tag{3.40}$$

Therefore, to compute the partial derivatives with respect to change of pose for the right halves, simply replace $\frac{b}{2}$ with $-\frac{b}{2}$ in Equations (3.36) through (3.39). The linearized C matrix when receiving i ranges with the Jackal's left beacon and j ranges with the Jackal's right beacon is then:

$$C = \begin{bmatrix} \frac{\partial r_1}{\partial x} & \frac{\partial r_1}{\partial y} & \frac{\partial r_1}{\partial z} & \frac{\partial r_1}{\partial \theta} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial r_{i+j}}{\partial x} & \frac{\partial r_{i+j}}{\partial y} & \frac{\partial r_{i+j}}{\partial z} & \frac{\partial r_{i+j}}{\partial \theta} \end{bmatrix} \tag{3.41}$$

3.4 Open-air accuracy check

To observe and quantify the level of accuracy of the Decawave devices using the default localization engine, a test was performed over the course of several hours. Eight units were configured as anchors, and one unit, in the middle of them, was configured as a tag, and connected via USB to a computer. From the Decawave shell interface, the command `les` was sent to the tag, which caused it to repeatedly report the anchors it was using for positioning, the distance to them, and its calculated position. Due to the ceiling having a split level, the bags on the Y-axis of the coordinate frame were at a uniform elevation that was different from the column of bags parallel to them. These two columns defined a plane that was not parallel to the floor, and so to satisfy the recommended conditions of the manufacturer, the bag in the middle, which contained the tag under test, was placed at a lower elevation. All of the hardware used for positioning the Decawave units used Imperial units, but the Decawaves perform their calculations in millimeter integers, so the values were converted.

Decawave units were connected to portable power banks and placed upright inside grommeted pouches, which were then suspended from the laboratory ceiling at measured heights and spacing as shown in Figure 3. The bags were hung from metal clips that attached to the ceiling tile support beams, which were spaced in Imperial units of measure. The beacons on the outside columns were configured as anchors, and a beacon on the inside was connected to a computer through a USB connection. The results are shown in Table 2 and Table 3.



Figure 3: Suspension of DWM modules from ceiling for open-air testing. Modules used in open-air test tabulated in this thesis have been annotated.

Table 2: Statistics of Open-Air Test by Anchor

Beacon ID:	C4A2 (Green1)	9932 (Blue1)	4D00 (Purple1)	CF04 (Black1)	CB82 (Green2)	9D0B (Red1)	C013 (Teal1)
Count of Measurements Recorded:	15192	15274	4059	3339	13113	5650	9236
Average Distance Measured (mm)	1052	1101	1780	2806	1654	3128	1718
Average Error (mm)	-83	-170	18	117	-12	379	-44
Max Distance Measured (mm)	1170	1270	2050	3350	2350	3520	2160
Max Error Measured (mm)	36	-1	289	661	685	770	399
Min Distance Measured (mm)	930	920	1650	2450	1550	2520	1550
Min Error Measured (mm)	-204	-351	-111	-239	-115	-230	-211
Standard deviation of measurement (mm)	32	47	37	52	30	95	51
True Distance (mm)	1134	1271	1761	2689	1665	2750	1761
X Position (mm)	0	0	1219	2438	-1219	2438	-1219
Y Position (mm)	0	2324	2324	0	0	2324	2324
Z Position (mm)	0	102	102	0	0	102	102

Table 3: Statistics of Localization Engine Calculations for Tag during Open-Air Test

Axis:	X	Y	Z
True Position (mm)	0	1105	-257
Average Position Measured (mm)	-99	1117	228
Standard Deviation (mm)	238	78	172
Count of Measurements Recorded:	15603	15603	15603

In addition to the range accuracy levels, performing this test revealed that although the Decawave units calculate values in millimeters, the firmware's Bluetooth user interface allowed the user to specify anchor positions only to the nearest centimeter. Also, the position information reported from the DWM's shell had some inconsistencies that made it difficult to parse. See the Operational Experience in Section 3.9 of this document for details.

3.5 Through-wall accuracy check

A test was conducted to determine how the accuracy of the Decawave modules changed when their signals propagated through walls and between floors of a building. With special approval from the Office of Campus Emergency Management and the University of Illinois Police Department, high-accuracy floor maps of the Transportation Building in Urbana, Illinois, were obtained. These maps were used to help accurately measure the placement of DWM units near landmarks in the building in separate rooms from the laboratory, both on the same floor, and on the floor above. DWM beacons were configured as anchors and hung throughout the building, programmed with their coordinates according to the lab's coordinate frame. Another DWM was programmed as a tag and was carried by the experimenter as they walked around the building, observing the accuracy of measurements on the Android tablet. However, when the experimenter left the laboratory and moved one meter to the side to take the first reading, the tag was no longer able to see any of the anchors in the laboratory, and was unable to provide a position estimate for itself. The lab's walls are cement blocks, one block thick, and this was enough to prevent the tag from obtaining sufficient positioning signal. This was not as expected, as Nekoogar [10] states UWB has superior penetration properties through a variety of materials, and is viable for through-wall communications. Those looking to implement this localization system on full-scale construction equipment will have to consider whether the signals of beacons will be blocked by the bodies of the vehicles on which they are mounted.

3.6 Spring measurement test in simulation

As an early test of the spring measurement code, a test was conducted in simulation. A ROS node was set up to broadcast the DWM range measurements that should be expected if a Jackal were equipped with one anchor and one tag DWM and positioned near other anchors and tags at known locations, as shown in Figure 4. The distance between beacons was calculated exactly, and then Gaussian white noise was added to the ideal measurements. The convergence threshold of the motion metric was set to 1 unit; that is, the spring measurement would announce its answer immediately after an iteration such that for the pose of each agent, the change in pose fit the criteria in Equation (3.42).

$$\sqrt{(x_{new} - x_{old})^2 + (y_{new} - y_{old})^2 + (z_{new} - z_{old})^2} \; + |\theta_{\text{new}} - \theta_{\text{old}}| < 1 \tag{3.42}$$

The spring constant was set to $0.001 \frac{N}{m}$. The starting position was varied from test to test, from a few millimeters of error, to meters of error, to kilometers of error. When the variance of the Gaussian white noise was low, the measurement returned an appropriate answer given the noisy measurement, even when the initial guess was off by several kilometers. However, for very large starting errors, the calculation would sometimes take a significant period of time to finish, on the order of a second. Because the simulated measurements were being broadcast at a rate of 10 Hz, this resulted in the spring measurement code appearing to freeze until it caught up in calculations, and then worked as expected afterwards. This behavior led to inserting a secondary stop condition for the spring measurement - if the calculation should take longer than 0.09 seconds, the value should be returned as is, so that the position measurement would be available at a rate of 10 Hz. The reasoning is that, even though the value may not have yet reached its final measurement location, it provided a more accurate position estimate than the uncorrected initial guess. By feeding that estimate back into the system that provided process updates and updated state estimates for the next iteration, the starting point of the next iteration would be close to where the measurement code left off on the previous iteration, and will continue to improve the estimate based on the latest measurements. This choice was observed to lead to smooth convergence of the spring measurement in simulation.

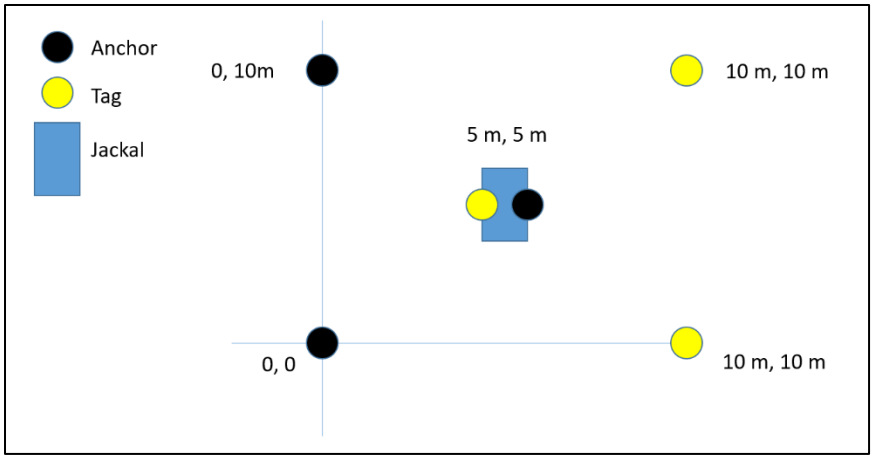


Figure 4: Spring measurement simulation environment.

3.7 Linear calibration of DWM ranges

The manufacturer of the DWMs provided a location in write-once memory where calibration information could be stored. Sources of error in DWM measurement include clock drift, frequency drift, antennae delays, and nonlinear effects of temperature and received signal strength. As the author was not yet confident enough in their understanding of the firmware to permanently write to the memory location where calibration data was stored, a zero-offset and scale correction was applied to ranges after measurement instead. To determine the correction that should be applied for each unit, five tripods were set up in a hallway as shown in Figure 5 such that five DWM beacons were in view of each other at known distances without interference. Beacons were rotated into the role of the tag, given a one-minute warmup time, and then their range data to the anchor beacons was recorded over durations of a minimum of three minutes. For each tag, a linear regression was applied to map the reported distances to the true distances. During this calibration, two DWMs stopped responding to Bluetooth commands, as detailed in Section 3.9.1. For most of the tags, a simple linear offset greatly reduced the median error in measurements, as the example in Table 4 shows.

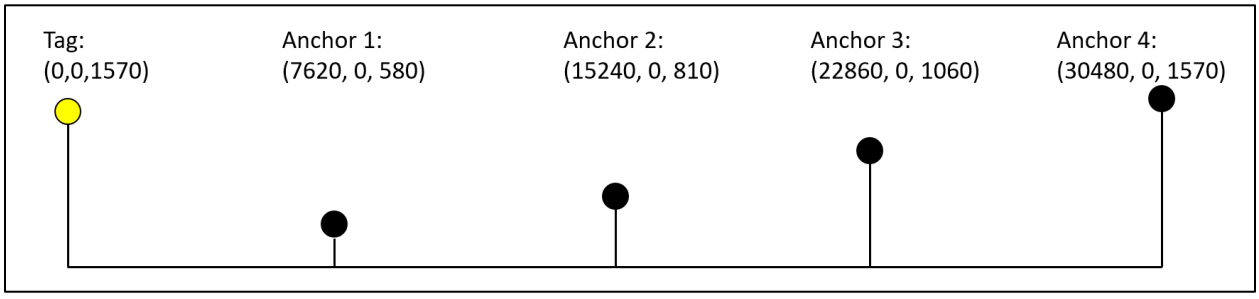


Figure 5: Placement of DWMs during linear calibration (millimeter units)

Table 4: Example of Linear Calibration Results for a DWM

	DWMCB82 Before Calibration	DWMCB82 After Calibration
Calibration Slope	N/A	1.000118
Calibration Offset	N/A	252.7607
Median Error (mm)	15568	-2.56199
Mean Error (mm)	18990	0
Standard Deviation of error	8389.826	43.28683

3.8 Outdoor tests

Five beacons were configured as anchors and placed on tripods on a semi-level surface as shown in Figure 6. The Jackal had two Decawave modules configured as tags and mounted on its left and right side, as shown in Figure 2. The Jackal was placed at initial starting position:

$$\begin{bmatrix} x \\ y \\ z \\ \theta \end{bmatrix} = \begin{bmatrix} -1220 \\ 610 \\ 0 \\ \frac{\pi}{2} \end{bmatrix} \tag{3.43}$$

The Jackal was driven around while the data from the IMU, wheel encoders, and Decawave modules was captured over the ROS network through the rosbag program. The data of those runs was then used to debug and evaluate the filter code. In the test run shown in Figure 6, the Jackal was driven in a cross pattern and returned to the starting position. The error in the position estimate calculated by the filters at the robot's end pose is shown in Table 5. The simple dead reckoning calculation, due to wheel slippage and model error, depicts the Jackal travelling beyond the starting position, and even beyond the beacon placed at the origin of the coordinate system. The Kalman filter using calibrated range values and least squares error positioning correctly showed the Jackal returning to its starting location, with a final error of 260 mm. The Kalman filter using the default Decawave firmware's systems is shown in the top right corner of Figure 6. Its total end pose error was only 29 cm worse than that of the Kalman filter using calibrated ranges and LSE, but the path history shows it did not closely match the Jackal's path as it approached Tripod B in the lower left corner of the plot. The error appears to have

approached 1 m in amplitude, but no precise measurements were captured. For future evaluations of the filters, a more descriptive ground truth metric than end pose should be used, such as using a motion capture system to record the entire path. The Kalman filter using the spring mesh calculation for the measurement input, shown in the lower left corner of Figure 7, performed so poorly that it cannot be said to have tracked the Jackal's position.

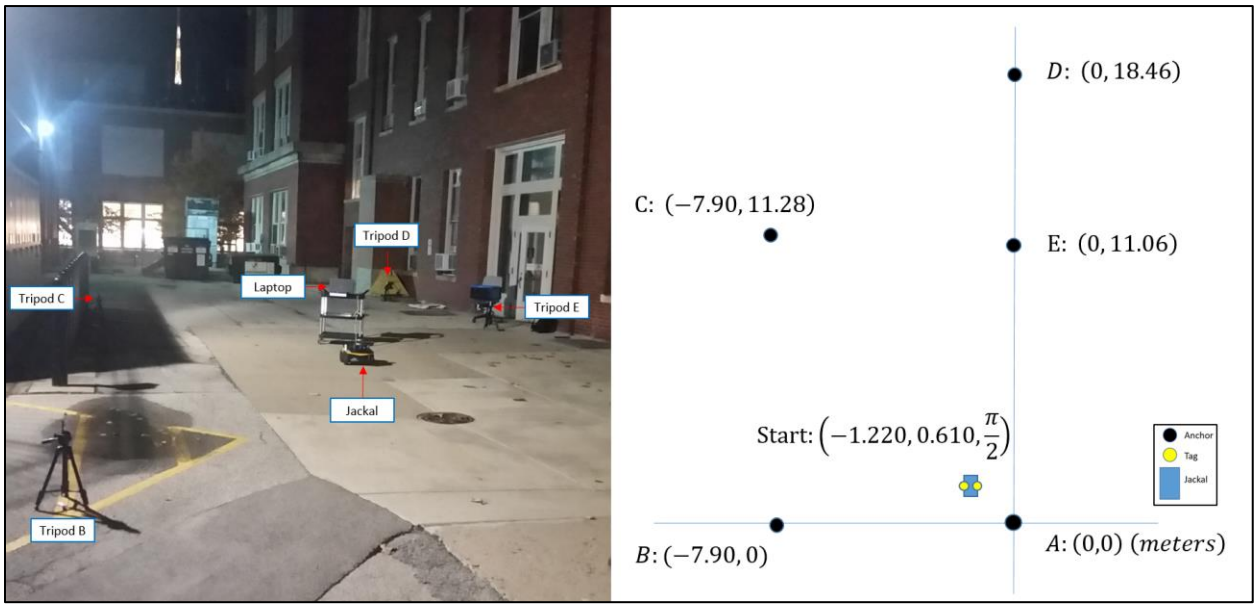


Figure 6: Left: The outdoor environment in which the Jackal was operated to test its filters. Right: Map of the beacon placements and Jackal starting pose.

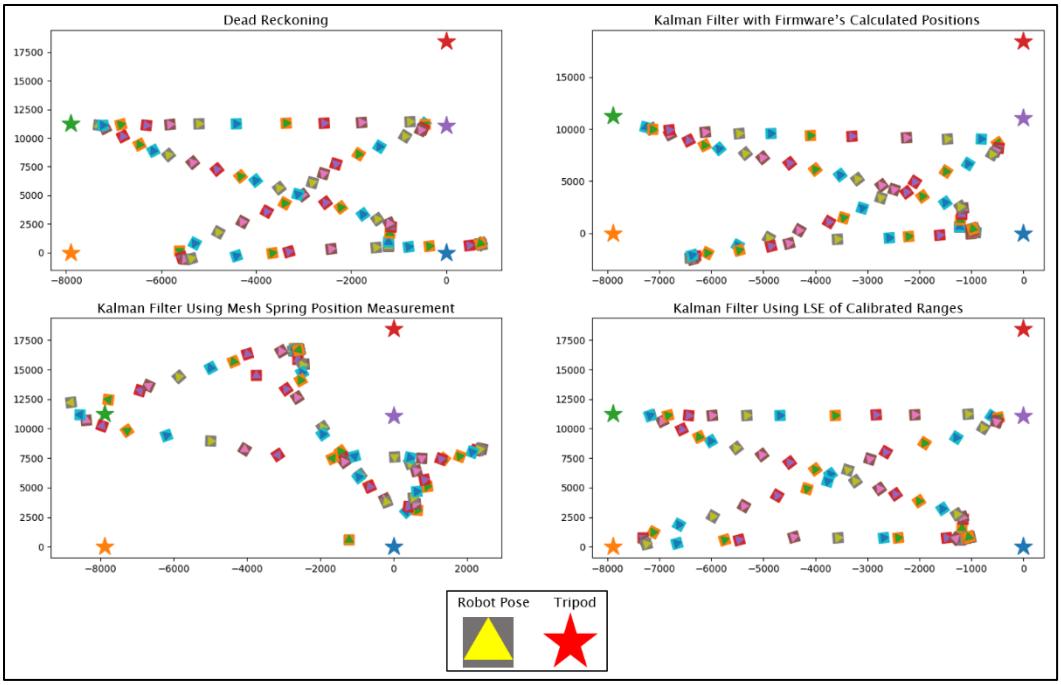


Figure 7: Performance of filters given same sensor data (units of millimeters).

Table 5: End Pose Error After Outdoor Localization Test

	Dead Reckoning	EKF with Firmware's Poses	EKF with Mesh Spring Measurement	EKF with LSE
Distance from Ground Truth	2024 mm	289 mm	16122 mm	260 mm
Δx	2018 mm	247 mm	-1390 mm	165 mm
Δy	159 mm	-150 mm	16062 mm	201 mm
$\Delta \theta$	16°	14°	104°	18°

Dramatic error in orientation seems to be a major contributor to the spring measurement's error. Error in orientation quickly results in error in position when the process updates occur. Samples of DWM data taken without IMU and encoder information, played back repeatedly to the filter, resulted in the filter measuring appropriate locations. However, the trajectory the estimates take during the correction process may be partially to blame for the poor performance. When the mesh measurement times out before it finishes calculation, the intermediate result, which is passed as a measurement to the Kalman filter, is not intuitive to predict. Figure 8 shows a simplified example of the behavior.

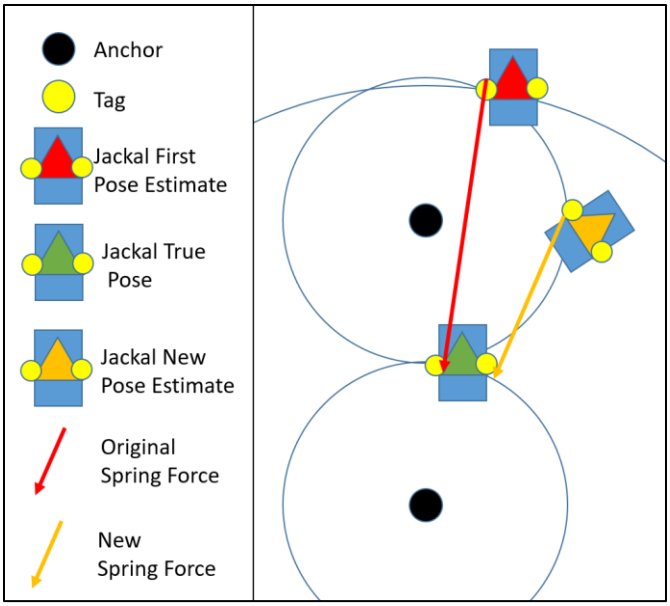


Figure 8: Illustration of partial position correction with spring measurement approach.

In the example shown in Figure 8, the measurements would eventually result in correcting the position of the Jackal's state estimate by moving it downward and to the left, while maintaining the same orientation. However, using the spring measurement approach, the anchor nearest the jackal pushes back against intermediate state estimates that would place it too close to the anchor for the given measurements. Therefore, the state estimate pivots the Jackal's body around, in this case clockwise, resulting in a temporary change in orientation estimate. If the measurement loop does not time out, it will correct the orientation after the Jackal's body has moved nearer the goal region. However, if the measurement loop is interrupted, it would output an erroneous orientation. Then, subsequent process updates would result in using that erroneous orientation to compute erroneous positions x and y.

Attempts were made to correct this problem by increasing the virtual spring constant. When the spring constant was very large, the position estimate of the robot oscillated violently. Improvement of the spring measurement approach is still in progress as of the time of this publication.

3.9 Operational experience

This section describes equipment failures encountered during work and advice to avoid potential pitfalls.

3.9.1 Failures of Decawave units

In one of the tests performed, all of the Decawave units were suspended from the ceiling and data had been recorded for several hours, when it was found that there was an error in the coordinate positions given to some the units, and they needed to be re-assigned positions through the Bluetooth interface. One of the units that had been working successfully stopped responding to Bluetooth commands. Its power bank was confirmed to have plenty of charge, and was capable of powering other units. Power cycling did not fix the problem with the unit. The unit continued sending out a strong Bluetooth signal, according to the manufacturer's Android application. The unit remained visible to other Decawave units through the UWB communication protocol and was able to communicate when plugged into a PC, but refused to acknowledge Bluetooth commands. Firmware from the manufacturer was downloaded onto a PC and flashed onto the malfunctioning unit, but it still failed to communicate through Bluetooth. This problem persisted for several days, and then the unit was set aside for over a month, unused. After that time, the unit was powered on for additional troubleshooting attempts, but there was no longer a need – the unit began communicating over Bluetooth, and was able to have its firmware updated to match that of the other units through the Android application. The cause of communications loss is currently unknown, as the unit had been suspended in the air, away from any anticipated hazards.

While collecting calibration data on a later date, two other DWM units similarly stopped responding to Bluetooth commands. In addition, while they responded correctly to the TLV command `dwm_loc_get`, they returned an error code indicating they did not recognize the TLV command `dwm_nodeid_get`, which is a command all DWM modules should understand, according to the API guide from Decawave [11].

3.9.2 Behavior of a DWM tag when it loses sight of all anchors

It has been discovered that when a DWM tag has been ranging with a set of anchors, and all of the anchors are disconnected from their power supply, sending it the TLV command `dwm_loc_get` will result in it continuing to report the distance to the last anchor it saw, even though it has lost contact with that anchor. This continues until more anchors are supplied with power, and the tag identifies new anchors. In writing a localization system that depends on UWB beacons, either the firmware must be modified to prevent this behavior, or a check should be performed on received range messages to ensure they are not identical to previous messages received.

3.9.3 Unexpected behavior of DWM shell

During the open-air accuracy test, the DWM's shell feature was given the `les` command, and it responded by sending localization information through the UART, which was written to a file by the listening computer, via PuTTY. However, over the course of several hours, the messages received were inconsistent. Some messages were interrupted by the message that was meant to

follow it, and that interrupting message will be interrupted by a third, and so on, until finally all of the ends of the messages were sent at once. This led to long sequences of one-beacon messages, two-beacon messages, and three-beacon messages when, in fact, the tag was seeing four beacons at the same time. See Appendix A for a sample of such output. Because of this behavior, the TLV interface of the DWMs was used instead of the shell interface.

3.9.4 Variation in range ability

During the calibration checks described earlier, most DWM devices were able to successfully range with all four anchors, but some of them failed to connect with the anchor placed at the 30.48 meter (100 feet) mark. In addition, sometimes an operating DWM tag would fail to connect with an anchor even when the anchor was close (within two meters). Power cycling the anchor and tag remedied this connection failure.

3.9.5 False information reported by DWMs

During the outdoor tests, after running for about 20 minutes, two of the five DWM modules falsely reported their ID as 0x0000, and gave extraordinarily large range values to the tags, corresponding to a distance multiple kilometers away. The false distances were only observed through the TLV interface. The Android Bluetooth application showed the anchors reporting their correct names and locations. Power cycling the DWMs corrected the problem.

3.9.6 Problems when using portable power banks

The Beston power banks were advertised to provide 5 V and 2.1 A through each of their dual USB ports. However, when the Raspberry Pi 3b+ was powered by these power banks, it provided a low voltage warning. A digital USB multimeter was used to check the actual voltage and current delivered by a fully charged power bank. When the Raspberry Pi 3b+ was running, the voltage dropped to 4.90 V while current draw was only 430 mA. To reduce the likelihood that an undervoltage problem would impact test results, all of the outdoor experiments were performed with the DWMs connected to the laptop computer, which was pushed behind the Jackal on a cart.

A second concern is that the Beston power banks sometimes turned themselves off after a few seconds when powering DWMs. Beston Power was contacted through their Amazon store account, and asked if there was a minimum current draw required to keep the power banks on. They stated the minimum current is 1 A, and the working voltage range is 3.7 V - 5 V. This 1 A minimum is greater than the largest current draw observed from the Raspberry Pi and two DWMs combined. Therefore, the author does not recommend the use of the Beston power bank to power the DWMs in future implementations.

3.9.7 Software conflicts

Many common packages used for ROS 1 were written in Python 2, and Python 2 is scheduled to be deprecated on January 1st, 2020 [15]. ROS 2, on the other hand, is still in development [16].

Because of this situation, the work described in this thesis was based on ROS 1, but some of the code was written in Python 3. One component used extensively in many ROS packages is TF, a transform library which allows users to keep track of multiple coordinate frames over time. Unfortunately, the binary installation of TF for Python 2 throws errors when an attempt is made to run it in Python 3. Recompiling it from source should allow it to be used with Python 3; however, the author was wary to do so, uncertain of what effects would occur in other ROS packages. Instead, a work-around was to call custom rospy nodes using the standard command line instruction `python <filename.py>` in lieu of using the rosrun command, or a roslaunch file. Even without calling rosrun, ROS nodes can still be generated.

When simulating the Jackal in Gazebo using the code provided by the manufacturer, the following warning was received:

```
[jackal1/urdf_spawner-8] process has died
```

The user may not notice this warning, as the Jackal will spawn anyway, and will respond to commands. However, the URDF spawner process dying is a problem that needs to be addressed if one is to simulate multiple Jackals under multiple namespaces. In future simulation work, the author intends to troubleshoot the reason the spawner dies in the given code, so that large numbers of Jackals may be simulated using the localization system.

3.10 Code publication

The code used for the localization engine is a work in progress, and not yet ready to be shared publicly. As the code becomes available, it will be published to the GitHub repository at https://github.com/JamesPKeiller/Keiller_MS_Thesis.

4 Conclusion

This thesis documented the progress toward a localization system for autonomous ground vehicles based on ultra-wideband beacons. The error in distance beacons was examined, and a linear calibration adjustment was found to provide an improvement in range estimates. Operational experience with the hardware selected for experiments was documented, and this increased understanding of the behavior of this UWB equipment facilitates future experiments. A script that communicates with DWMs and publishes their information to a ROS network was tested successfully. Several localization techniques were implemented, including an experimental expansion of a mesh localization technique. While testing with real beacons and a real robot in an outdoor environment, it was found that further improvement is needed, as the extended Kalman filters all exhibited error of at least 26 cm in end pose calculations, and the mesh technique could not track the robot as currently implemented.

For future work, development of failure detection and recovery for nodes that communicate with DWM beacons will be useful in developing the localization system. The findings related to failures of DWM modules should be compared against the operational experiences of others on the Decawave forums, to better understand the cause and remedies.

With respect to the localization system's algorithms, the modified spring mesh measurement technique needs additional testing with simple real-world experimental setups, similar to the earlier simulations. This testing will determine if the current behavior is a result of a bug in software implementation, or the algorithm itself. The extended Kalman filter using the range-only measurement model, as derived in Equations (3.34) through (3.41), should be implemented in code and evaluated against the other filters. Finally, in the future it may become necessary to update the Jackal's state vector to include pitch and roll, so that the localization system can function appropriately for environments other than flat ground.

References

[1] A. Howard, M. J. Mataric, and G. Sukhatme, "Relaxation on a mesh: A formalism for generalized localization," in *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001, pp. 1055–1060.

[2] S. Tully, G. Kantor, and H. Choset, "Leap-frog path design for multi-robot cooperative localization," *Field and Service Robotics*, vol. 62, pp. 307–317, 2010.

[3] A. Buchan, "Towards cooperative SLAM for low-cost biomimetic robots," Ph.D. dissertation, University of California, Berkeley, CA, 2017.

[4] A. Benini, A. Mancini, A. Marinelli, and S. Longhi, "A biased extended Kalman filter for indoor localization of a mobile agent using low-cost IMU and UWB wireless sensor network," *IFAC Proceedings Volumes*, vol. 45, no. 22, pp. 735-740, 2012.

[5] L. Zhang, H. Zhang, X. R. Cui, and T. A. Gulliver, "Ultra wideband indoor positioning using Kalman filters," *Advanced Materials Research*, vol. 433–440, pp. 4207–4213, 2012.

[6] "ROS.org — Powering the world's robots." Accessed on: Nov. 14, 2019. [Online]. Available: http://www.ros.org/

[7] Clearpath Robotics, *Jackal UGV - Small Weatherproof Robot*, Accessed on: Nov. 14, 2019. [Online]. Available: https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/

[8] R. Siegwart and I. Nourbakhsh, "Perception," in *Introduction to Autonomous Mobile Robots*. Cambridge, MA: MIT Press, 2004, ch. 4, pp. 79–158.

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2014, pp. 1–11.

[10] F. Nekoogar, "Introduction to ultra-wideband communications," in *Ultra-Wideband Communications: Fundamentals and Applications*. Upper Saddle River, NJ: Prentice Hall Press, 2005, ch. 1, pp. 1-44.

[11] *Dwm1001 Firmware Application Programming Interface (API) Guide*, Decawave, Dublin, Ireland, 2019. Accessed on: Jun. 3, 2019. [Online]. Available: https://www.decawave.com/product-documentation/

[12] G. Blewitt, "Basics of the GPS technique: Observation equations," in *Geodetic Applications of GPS,* Bastad, Sweden: Swedish Land Survey, 1997, pp. 10-54.

[13] Y. Bulut, "Applied Kalman filter theory," Ph.D. dissertation, Northwestern University, Boston, MA, 2011.

[14] H. M. Choset et al., "Kalman filtering," in *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2009, pp. 290–300.

[15] T. O'Brien, *Sunsetting Python 2 Support*, 2019. Accessed on: Nov. 16, 2019. [Online]. Available: https://www.python.org/doc/sunset-python-2/

[16] "ROS 2 Overview" Accessed on: Nov. 16, 2019. [Online]. Available: https://index.ros.org/doc/ros2/

Appendix A: Sample Output of DWM Shell

When the UART shell provided by the default Decawave firmware was given the " les" command, a request for the localization information visible to a tag, the result was sometimes of an inconsistent format that would be difficult to parse reliably. This necessitated writing a node to communicate with the TLV interface instead. To illustrate the inconsistency, a sample of the output from the open-air test in Section 3.4 is shown below.

```
C4A2[0.00,0.00,0.00]=1.06          9932[0.00,2.32,0.10]=1.09          CB82[-
1.22,0.00,0.00]=1.67          C013[-1.22,2.32,0.10]=1.76          le_us=4638
est[0.16,1.15,0.14,72]
C4A2[0.00,0.00,0.00]=1.00          9932[0.00,2.32,0.10]=1.12          CB82[-
1.22,0.00,0.00]=1.63          C013[-1.22,2.32,0.10]=1.68          le_us=4821
est[0.16,1.14,0.21,74]
C4A2[0.00,0.00,0.00]=1.08          9932[0.00,2.32,0.10]=1.09          CB82[-
1.22,0.00,0.00]=1.69          C013[-1.22,2.32,0.10]=1.66          le_us=4272
est[0.15,1.14,0.22,72]
C4A2[0.00,0.00,0.00]=1.02          9932[0.00,2.32,0.10]=1.07          CB82[-
1.22,0.00,0.00]=1.67          C013[-1.22,2.32,0.10]=1.74          le_us=4913
est[0.18,1.13,0.24,77]
C4A2[0.00,0.00,0.00]=1.02          9932[0.00,2.32,0.10]=1.11          CB82[-
1.22,0.00,0.00]=1.66          C013[-1.22,2.32,0.10]=1.73          le_us=4852
est[0.19,1.12,0.29,76]
C4A2[0.00,0.00,0.00]=1.05          9932[0.00,2.32,0.10]=1.03          CB82[-
1.22,0.00,0.00]=1.65          C013[-1.22,2.32,0.10]=1.67          le_us=5096
est[0.19,1.12,0.31,75]
C4A2[0.00,0.00,0.00]=1.05          9932[0.00,2.32,0.10]=1.12          CB82[-
1.22,0.00,0.00]=1.65          C013[-1.22,2.32,0.10]=1.75          le_us=4394
est[0.17,1.12,0.29,72]
C4A2[0.00,0.00,0.00]=1.05          9932[0.00,2.32,0.10]=1.10          CB82[-
1.22,0.00,0.00]=1.63          C013[-1.22,2.32,0.10]=1.70          le_us=4364
est[0.15,1.12,0.27,71]
C4A2[0.00,0.00,0.00]=1.05          9932[0.00,2.32,0.10]=1.10          CB82[-
1.22,0.00,0.00]=1.65          C013[-1.22,2.32,0.10]=1.67          le_us=4302
est[0.14,1.12,0.25,71]
C4A2[0.00,0.00,0.00]=1.12          9932[0.00,2.32,0.10]=1.09          CB82[-
1.22,0.00,0.00]=1.67          C013[-1.22,2.32,0.10]=1.71          le_us=4394
est[0.13,1.14,0.12,73]
```

```
C4A2[0.00,0.00,0.00]=1.03          9932[0.00,2.32,0.10]=1.11          CB82[-
1.22,0.00,0.00]=1.63          C013[-1.22,2.32,0.10]=1.71          le_us=4364
est[0.13,1.13,0.12,71]
C4A2[0.00,0.00,0.00]=1.07          9932[0.00,2.32,0.10]=1.06          CB82[-
1.22,0.00,0.00]=1.70  C013[-1.22,2.32,0.10]=1.72  le_us=4882  est[0.15,1.15,-
0.01,77]
C4A2[0.00,0.00,0.00]=1.10          9932[0.00,2.32,0.10]=1.16          CB82[-
1.22,0.00,0.00]=1.66          C013[-1.22,2.32,0.10]=1.72          le_us=4089
est[0.14,1.13,0.09,71]
C4A2[0.00,0.00,0.00]=1.05          9932[0.00,2.32,0.10]=1.10          CB82[-
1.22,0.00,0.00]=1.65          C013[-1.22,2.32,0.10]=1.70          le_us=4394
est[0.13,1.13,0.11,72]
C4A2[0.00,0.00,0.00]=1.08          9932[0.00,2.32,0.10]=1.02          CB82[-
1.22,0.00,0.00]=1.71  C013[-1.22,2.32,0.10]=1.78  le_us=5096  est[0.15,1.15,-
0.00,76]
C4A2[0.00,0.00,0.00]=1.05          9932[0.00,2.32,0.10]=1.11          CB82[-
1.22,0.00,0.00]=1.62          C013[-1.22,2.32,0.10]=1.68          le_us=4150
est[0.13,1.14,0.06,71]
C4A2[0.00,0.00,0.00]=0.99          9932[0.00,2.32,0.10]=1.16          CB82[-
1.22,0.00,0.00]=1.67          C013[-1.22,2.32,0.10]=1.71          le_us=4547
est[0.12,1.12,0.10,69]
C4A2[0.00,0.00,0.00]=1.07          9932[0.00,2.32,0.10]=1.10          CB82[-
1.22,0.00,0.00]=1.63          C013[-1.22,2.32,0.10]=1.72          le_us=4425
est[0.12,1.13,0.15,72]
C4A2[0.00,0.00,0.00]=1.05          9932[0.00,2.32,0.10]=1.07          CB82[-
1.22,0.00,0.00]=1.67          C013[-1.22,2.32,0.10]=1.71          le_us=4882
est[0.14,1.12,0.22,76]
C4A2[0.00,0.00,0.00]=1.03          9932[0.00,2.32,0.10]=1.13          CB82[-
1.22,0.00,0.00]=1.64          C013[-1.22,2.32,0.10]=1.69          le_us=4302
est[0.13,1.12,0.22,71]
C4A2[0.00,0.00,0.00]=1.12          9932[0.00,2.32,0.10]=1.12          CB82[-
1.22,0.00,0.00]=1.63          C013[-1.22,2.32,0.10]=1.70          le_us=3845
est[0.10,1.11,0.22,68]
C4A2[0.00,0.00,0.00]=1.06          9932[0.00,2.32,0.10]=1.05          CB82[-
1.22,0.00,0.00]=1.66          C013[-1.22,2.32,0.10]=1.77          le_us=5096
est[0.12,1.13,0.06,75]
```

```
C4A2[0.00,0.00,0.00]=1.10          9932[0.00,2.32,0.10]=1.07          CB82[-
1.22,0.00,0.00]=1.67          C013[-1.22,2.32,0.10]=1.71          le_us=4425
est[0.12,1.14,0.00,73]
C4A2[0.00,0.00,0.00]=1.04          9932[0.00,2.32,0.10]=1.07          CB82[-
1.22,0.00,0.00]=1.68  C013[-1.22,2.32,0.10]=1.65  le_us=4821  est[0.13,1.16,-
0.06,74]
C4A2[0.00,0.00,0.00]=1.04          9932[0.00,2.32,0.10]=1.12          CB82[-
1.22,0.00,0.00]=1.61          C013[-1.22,2.32,0.10]=1.74          le_us=4394
est[0.12,1.14,0.01,72]
9D0B[2.44,2.32,0.10]=3.10
9D0B[2.44,2.32,0.10]=3.12
9D0B[2.44,2.32,0.10]=3.12
9D0B[2.44,2.32,0.10]=3.15
9D0B[2.44,2.32,0.10]=3.20
C4A2[0.00,0.00,0.00]=1.01          9932[0.00,2.32,0.10]=1.03          C013[-
1.22,2.32,0.10]=1.70 le_us=1281 est[0.16,1.14,0.09,81]
C4A2[0.00,0.00,0.00]=1.05          9932[0.00,2.32,0.10]=1.06          CB82[-
1.22,0.00,0.00]=1.64 le_us=1098 est[0.15,1.14,0.19,75]
C4A2[0.00,0.00,0.00]=0.99          9932[0.00,2.32,0.10]=1.09          CB82[-
1.22,0.00,0.00]=1.63 le_us=1129 est[0.16,1.13,0.25,75]
C4A2[0.00,0.00,0.00]=1.02          9932[0.00,2.32,0.10]=1.09          CB82[-
1.22,0.00,0.00]=1.63 le_us=1129 est[0.16,1.12,0.32,74]
C4A2[0.00,0.00,0.00]=1.04          9932[0.00,2.32,0.10]=1.11          CB82[-
1.22,0.00,0.00]=1.64 le_us=915 est[0.14,1.12,0.28,95]
C4A2[0.00,0.00,0.00]=1.05          9932[0.00,2.32,0.10]=1.12          CB82[-
1.22,0.00,0.00]=1.64 le_us=915 est[0.13,1.12,0.28,91]
9D0B[2.44,2.32,0.10]=3.11
9932[0.00,2.32,0.10]=1.02
CB82[-1.22,0.00,0.00]=1.71
C013[-1.22,2.32,0.10]=1.71
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.75
C013[-1.22,2.32,0.10]=1.81
```

```
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.83
CB82[-1.22,0.00,0.00]=1.69
CB82[-1.22,0.00,0.00]=1.70 9D0B[2.44,2.32,0.10]=3.11
4D00[1.22,2.32,0.10]=1.81
9D0B[2.44,2.32,0.10]=3.11
4D00[1.22,2.32,0.10]=1.74
9D0B[2.44,2.32,0.10]=3.10
4D00[1.22,2.32,0.10]=1.70 C013[-1.22,2.32,0.10]=1.85
C013[-1.22,2.32,0.10]=1.76 4D00[1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.78 4D00[1.22,2.32,0.10]=1.81
C013[-1.22,2.32,0.10]=1.78 4D00[1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.79 4D00[1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.74 4D00[1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.76 4D00[1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.80 4D00[1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.77 4D00[1.22,2.32,0.10]=1.75
C013[-1.22,2.32,0.10]=1.78 4D00[1.22,2.32,0.10]=1.83
C013[-1.22,2.32,0.10]=1.78 4D00[1.22,2.32,0.10]=1.81
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.72
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.81
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.75
C013[-1.22,2.32,0.10]=1.76
```

```
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.82
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.75
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.81
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.72
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.71
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.75
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.70
```

```
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.75
C013[-1.22,2.32,0.10]=1.71
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.71
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.81
C013[-1.22,2.32,0.10]=1.69
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.81
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.70
C013[-1.22,2.32,0.10]=1.72
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.81
C013[-1.22,2.32,0.10]=1.78
```

```
C013[-1.22,2.32,0.10]=1.75
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.71
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.72
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.72
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.75
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.72
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.75
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.77
```

```
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.75
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.80
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.81
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.73
C013[-1.22,2.32,0.10]=1.75
C013[-1.22,2.32,0.10]=1.75
C013[-1.22,2.32,0.10]=1.82
C013[-1.22,2.32,0.10]=1.74
C013[-1.22,2.32,0.10]=1.79
C013[-1.22,2.32,0.10]=1.76
C013[-1.22,2.32,0.10]=1.77
C013[-1.22,2.32,0.10]=1.78
C013[-1.22,2.32,0.10]=1.77          C4A2[0.00,0.00,0.00]=1.05          CB82[-
1.22,0.00,0.00]=1.64          9932[0.00,2.32,0.10]=1.17          le_us=4180
est[0.13,1.11,0.30,72]
C4A2[0.00,0.00,0.00]=1.03                          9932[0.00,2.32,0.10]=1.04
CF04[2.44,0.00,0.00]=2.78        CB82[-1.22,0.00,0.00]=1.63        le_us=5676
est[0.02,1.12,0.31,61]
C4A2[0.00,0.00,0.00]=1.05                          9932[0.00,2.32,0.10]=1.09
CF04[2.44,0.00,0.00]=2.81    CB82[-1.22,0.00,0.00]=1.63    le_us=5462    est[-
0.06,1.12,0.35,62]
C4A2[0.00,0.00,0.00]=1.01          9932[0.00,2.32,0.10]=1.07          CB82[-
1.22,0.00,0.00]=1.65     C013[-1.22,2.32,0.10]=1.62     le_us=4852     est[-
0.01,1.12,0.38,74]
C4A2[0.00,0.00,0.00]=1.06          9932[0.00,2.32,0.10]=1.01          CB82[-
1.22,0.00,0.00]=1.68          C013[-1.22,2.32,0.10]=1.68          le_us=5096
est[0.04,1.14,0.21,76]
C4A2[0.00,0.00,0.00]=1.07                          9932[0.00,2.32,0.10]=1.14
CF04[2.44,0.00,0.00]=2.85    CB82[-1.22,0.00,0.00]=1.62    le_us=4943    est[-
0.00,1.15,0.13,66]
```

```
C4A2[0.00,0.00,0.00]=1.03          9932[0.00,2.32,0.10]=1.01          CB82[-
1.22,0.00,0.00]=1.68          C013[-1.22,2.32,0.10]=1.72          le_us=5371
est[0.06,1.16,0.03,77]
C4A2[0.00,0.00,0.00]=1.10                          9932[0.00,2.32,0.10]=1.07
CF04[2.44,0.00,0.00]=2.76        CB82[-1.22,0.00,0.00]=1.66        le_us=5310
est[0.01,1.17,-0.01,65]
9932[0.00,2.32,0.10]=1.15                          C4A2[0.00,0.00,0.00]=1.08
CF04[2.44,0.00,0.00]=2.75    C013[-1.22,2.32,0.10]=1.70    le_us=3631    est[-
0.03,1.15,0.08,82]
C4A2[0.00,0.00,0.00]=1.03          9932[0.00,2.32,0.10]=1.11          CB82[-
1.22,0.00,0.00]=1.66     C013[-1.22,2.32,0.10]=1.67     le_us=4333     est[-
0.00,1.14,0.11,71]
C4A2[0.00,0.00,0.00]=1.12          9932[0.00,2.32,0.10]=1.15          CB82[-
1.22,0.00,0.00]=1.66          C013[-1.22,2.32,0.10]=1.64          le_us=3845
est[0.00,1.14,0.20,71]
C4A2[0.00,0.00,0.00]=1.02                          9932[0.00,2.32,0.10]=1.08
CF04[2.44,0.00,0.00]=2.77    CB82[-1.22,0.00,0.00]=1.65    le_us=5645    est[-
0.07,1.13,0.26,62]
9D0B[2.44,2.32,0.10]=3.11
C4A2[0.00,0.00,0.00]=1.02 9D0B[2.44,2.32,0.10]=3.11
9D0B[2.44,2.32,0.10]=3.20 C4A2[0.00,0.00,0.00]=0.99
9D0B[2.44,2.32,0.10]=3.14 C4A2[0.00,0.00,0.00]=1.00
9D0B[2.44,2.32,0.10]=3.13 C4A2[0.00,0.00,0.00]=1.03
9D0B[2.44,2.32,0.10]=3.14 C4A2[0.00,0.00,0.00]=1.05
9D0B[2.44,2.32,0.10]=3.14 C4A2[0.00,0.00,0.00]=1.01
9D0B[2.44,2.32,0.10]=3.13 C4A2[0.00,0.00,0.00]=1.07
9D0B[2.44,2.32,0.10]=3.16 C4A2[0.00,0.00,0.00]=1.06
9D0B[2.44,2.32,0.10]=3.14 C4A2[0.00,0.00,0.00]=1.05
9D0B[2.44,2.32,0.10]=3.16 C4A2[0.00,0.00,0.00]=1.04
9D0B[2.44,2.32,0.10]=3.17 C4A2[0.00,0.00,0.00]=1.07
9D0B[2.44,2.32,0.10]=3.15 C4A2[0.00,0.00,0.00]=1.03
9D0B[2.44,2.32,0.10]=3.16 C4A2[0.00,0.00,0.00]=1.04
9D0B[2.44,2.32,0.10]=3.13 C4A2[0.00,0.00,0.00]=1.03
```

Appendix B: Sample Configuration File

Following is an example of a configuration file users can modify to add vehicles and tripods to the fleet without needing to modify the code.

```
# The parser will look for the file name active_configuration.txt.
# Save copies of the various configurations you use
# and rename the one you want to use to be active_configuration.txt.
# The parser ignores all text that comes after a hash sign.  Do not put
# white spaces in text strings.  The first word of a line describes what
# information that line carries with it. - James Keiller Fall 2019

# To properly disable robot bodies, you need to comment out the robot's
# body description AND the DWM assignments that reference the robot.

# Messages currently supported:
# DWM_ASSIGNMENT,DWM_ID,TAG/ANCHOR,BODY_ID,ROLE
#BODY_DESCRIPTION_2D,ARCHETYPE,BODY_ID,IMU_TOPIC_NAME,LINEAR_X,LINEAR_Y,LIN
EAR_Z,THETA
#BODY_DESCRIPTION_QUATERNION,ARCHETYPE,BODY_ID,IMU_TOPIC_NAME,LINEAR_X,LINE
AR_Y,LINEAR_Z,QUATERNION_X, QUATERNION_Y, QUATERNION_Z, QUATERNION_W
#DWM_CALIBRATION,DWM_ID,LINEAR_SLOPE,ZERO_OFFSET

BODY_DESCRIPTION_2D,JACKAL,JACKAL1,jackal_velocity_controller/odom,0,0,0,0
BODY_DESCRIPTION_2D,JACKAL,JACKAL2,jackal_velocity_controller2/odom,0,0,0,0
BODY_DESCRIPTION_2D,JACKAL,JACKAL3,jackal_velocity_controller3/odom,0,0,0,0
DWM_ASSIGNMENT,FAKE_ID,TAG,JACKAL1,LEFT
DWM_ASSIGNMENT,REAL_ID,ANCHOR,JACKAL1,RIGHT
DWM_CALIBRATION,CINDY,0.99,102

# ARCHETYPE options:
# Right now, only JACKAL and TRIPOD are implemented.  More will be added as
# needed

# Jackal options:
# ARCHETYPE should be JACKAL.  This selects the class in the python
# scripts.
# BODY_ID is what you want to call the Jackal.
# IMU_TOPIC_NAME is the ROS topic to subscribe to to receive IMU data
# the x,y,z theta, and quaternion info are for initial pose estimates.
# When assigning DWMs, the available roles are LEFT and RIGHT


# DWM_ASSIGNMENT options:
# TAG/ANCHOR should be either TAG or ANCHOR, depending upon the role of the
device
# BODY_ID matches the tag to one of the bodies specified by this config file.

# DWM_CALIBRATION options:
# If you haven't found the calibration data yet for your device, setting
# LINEAR_SLOPE to 1 and ZERO_OFFSET to 0 will be the same as running without
# attempting to adjust for error in that device.
# calibrated_measurement = LINEAR_SLOPE*original_measurement + ZERO_OFFSET
```

Appendix C: DWM Range Publisher Code

```
"""
dwmRangePublisher.py
Written September 2019 by James Keiller
Intended for use with Decawave DWM 1001 module through UART TLV interface
This script uses API calls as specified in the DWM1001 Firmware API Guide
5.3.10
Use command line arguments to specify the name of the usb serial port to use
(see myParser() function)
TODO: Expand script to allow position updates to be written to anchor nodes
Currently limited to Python 3.6+
"""
# Written September 2019 by James Keiller
# Intended for use with DWM 1001 module through UART TLV interface
# This script calls the dwm_loc_get API call as specified in the
# DWM1001 Firmware API Guide 5.3.10.
# It parses the information received to send over
# the ROS network.
# In the future, this script will be expanded to allow
# position updates to be written to anchor nodes
# Currently limited to Python 3.6+.  Use command line arguments
# to specify the name of the port (See myParser() function)


# region Import Statements
import serial  # use "pip install pyserial" if you have not already done so
import time
import argparse
import inspect
import rospy
from dwm_communication.msg import dwmRange, dwmPositionData
from std_msgs.msg import String, Header
# endregion


# region Setup Code
# defaultPortName can be overridden with command line arguments
```

```
# see helper function myParser() for command line arguments
defaultPortName = '/dev/ttyACM0'
# On linux:
# Use /dev/ttyACM0 or similar
# calling "dmesg | grep tty" and looking at the last line will show you the
last tty device you plugged in.
# On Windows, the port name may be 'COM9' or similar
# Use Device Manager, and look under the PORTS heading.

ser = None   # This will be the name of the handle to the serial port
# endregion



# region Constant Definitions
EXIT_SUCCESS = 0
EXIT_FAILURE = 1
# API Error codes
ERR_CODE_OK = bytes.fromhex("00")
# 1: unknown command or broken TLV frame
# 2: internal error
# 3: invalid parameter
# 4: busy
# 5: operation not permitted
# API Commands [Type, length]
DWM_POS_SET = bytes.fromhex("01 0d")                    # Used to set position.
Follow with position as 13 bytes
DWM_POS_GET = bytes.fromhex("02 00")                    #  Used  to  ask  for
position.
DWM_NODEID_GET = bytes.fromhex("30 00")          # Used to ask for 8-byte ID
of node.
DWM_CFG_GET = bytes.fromhex("08 00")                    #              Request
configuration information of DWM
DWM_LOC_GET = bytes.fromhex("0c 00")                    # Request for position
+ distances to anchors/tags
# Response codes
TLV_TYPE_DUMMY = bytes.fromhex("00")                    #  Reserved  for  SPI
dummy byte
```

```
TLV_TYPE_POS_XYZ = bytes.fromhex("41")          #      Response      position
coordinates x,y,z with q
TLV_TYPE_CFG = bytes.fromhex("46")                    #            Response:
configuration information of DWM
TLV_TYPE_RNG_AN_DIST = bytes.fromhex("48")      #  Response:  Ranging  anchor
distances
TLV_TYPE_RNG_AN_POS_DIST = bytes.fromhex("49")   # Response: Ranging anchor
distances and positions
TLV_TYPE_NODE_ID = bytes.fromhex("4E")          #    Response:    Node    ID.
Followed by a length of 8.



# Custom classes
class mydwmRange(dwmRange):
      """
      This is a sub-class of the dwmRange message class.  Creating a
      sub-class allows for the creation of an initializer and methods.
      """
      def __init__(self):
            self.header = Header()
            self.dwm_ID = "Not specified"
            self.position_is_valid = False
            self.x = 0
            self.y = 0
            self.z = 0
            self.position_quality = 0
            self.distance = 0
            self.distance_quality = 0


class mydwmPositionData(dwmPositionData):
      """
      This is a sub-class of the dwmPositionData class.
      Creating a sub-class allows for the creation of an
      initializer and methods
      """
      def __init__(self):
            self.header = Header()
```

```
            self.dwmID = "Not specified"
            self.isAnchor = False
            self.positionIsValid = False
            self.x = 0
            self.y = 0
            self.z = 0
            self.q = 0
            self.numRanges = 0
            self.ranges = []

      def updateLocationData(self, locData):
            """
            Updates the localization data for this message envelope.
            :param locData: The return value from getLocations().
            [position_data, distances_data]
            where position_data is of the form [x,y,z,q]
            and distances data is a list of lists in one of two forms:
            [[addr, d, dq],...] or
            [[addr, d, dq, x, y, z, q], ....]
            addr is  string
            :return: None
            """
            self.ranges = []
            self.positionIsValid = False
            number_of_anchored_ranges = 0
            [x, y, z, q] = locData[0]
            self.x = x
            self.y = y
            self.z = z
            self.q = q
            self.numRanges = len(locData[1])
            for i in range(self.numRanges):
                  frame = mydwmRange()
                  data = locData[1][i]
                  frame.dwm_ID = data[0]
                  frame.distance = data[1]
                  frame.distance_quality = data[2]
                  if len(data) == 7:
```

```
                        frame.x = data[3]
                        frame.y = data[4]
                        frame.z = data[5]
                        frame.position_quality = data[6]
                        frame.position_is_valid = True
                        number_of_anchored_ranges += 1
                  self.ranges.append(frame)
            # TODO: Update "valid position" logic for anchors.  This is just
for tags
            if number_of_anchored_ranges > 3:
                  if not self.isAnchor:
                        self.positionIsValid = True
            self.header.stamp = rospy.Time.now()
# endregion


# region Primary Functions
def main():
      global ser
      print("dwmPosGet started.")
      myPort = myParser()
      positionMessage = mydwmPositionData()  # Create a blank message frame
      # Establish serial port connection
      dwmConnected = False
      dwm_pub   =   rospy.Publisher('full_dwm_chatter',   mydwmPositionData,
queue_size=10)
      rospy.init_node('dwmTalker', anonymous=True)
      rate = rospy.Rate(1)  # 1 Hz
      while not rospy.is_shutdown():
            while dwmConnected is False:
                  try:
                        ser    =    serial.Serial(myPort,    baudrate=115200,
timeout=None)
                        print(ser)
                        print("Connection established.")
                        dwmConnected = True
                  except serial.SerialException:
```

```
                        print("Error in trying to connect to serial port
{}".format(myPort))
                        dwmConnected = False
                        time.sleep(0.01)
            # Find out device ID and whether it is an anchor or tag
            #try:
            device_identified = False
            while not device_identified:
                  try:
                        positionMessage.dwmID = get_node_ID().hex()
                        positionMessage.isAnchor = ask_if_anchor()
                        if positionMessage.isAnchor:
                              print("Connected to ANCHOR
{0}.".format(positionMessage.dwmID))
                        else:
                              print("Connected to TAG
{0}.".format(positionMessage.dwmID))
                        device_identified = True
                  except dwmError:
                        print("Exception: dwmError while dwmConnected is
False. Passing...")
                        pass
                  except serial.SerialException:
                        dwmConnected = False
                        print("dwm Disconnected. Trying to re-establish.")
                        time.sleep(0.5)
                        break
            while dwmConnected is True:
                  try:
                        locData = getLocations()
                        positionMessage.updateLocationData(locData)
                        rospy.loginfo(positionMessage)
                        dwm_pub.publish(positionMessage)
                        rate.sleep()
                  except dwmError:
                        print("Except: dwmError while dwmConnected is True.")
                        pass
                  except serial.SerialException:
```

```
                        dwmConnected = False
                        print("dwm Disconnected.  Trying to re-establish.")
                        time.sleep(0.5)


def getLocations():
      """
      Asks the Decawave unit for position information and reports response
      :rtype: Will return [position_data, distances_data]
      where position_data is of the form [x,y,z,q]
      and distances_data is a list in one of two forms:
      [[addr, d, dq],...] or
      [[addr, d, dq, x, y, z, q], ....]
      Note: The list may be empty, and will be as [].
      """
      #Ask Decawave for Position and distances
      temp = sendTLV(DWM_LOC_GET)
      if temp == EXIT_FAILURE:
            raise dwmRequestError("sendTLV returned with EXIT_FAILURE")
      # Receive confirmation of request / error code
      handle_confirmation_TLV_frame()
      # Read until get the position
      [typeTLV, length, value] = receiveTLV()  # Expect Position
      if length < 13:
            print("No position received.  Flushing buffer.")
            ser.reset_input_buffer()
            raise dwmUnexpectedMessage("getLocations- length < 13")
      else:
            # printTLV(typeTLV, length, value)
            position = parseTLV(typeTLV, length, value)
      [typeTLV, length, value] = receiveTLV()  # Expect Distances
      if length < 13:
            print("No distances received")
            distances = []
      else:
            # printTLV(typeTLV, length, value)
            distances = parseTLV(typeTLV, length, value)
      return [position, distances]
```

```
def ask_if_anchor():
      # Send configuration request
      global ser
      request = DWM_CFG_GET
      acknowledge = sendTLV(request)
      if acknowledge == EXIT_FAILURE:
            raise dwmTransmitError
      # Receive confirmation of request / error code
      handle_confirmation_TLV_frame()
      # Receive Configuration information
      [typeTLV, length, value] = receiveTLV()
      enforce_TLV_type(typeTLV, TLV_TYPE_CFG)
      anchor = parseTLV(typeTLV, length, value)
      return anchor


def get_node_ID():
      global ser
      request = DWM_NODEID_GET
      acknowledge = sendTLV(request)
      if acknowledge == EXIT_FAILURE:
            print("send TLV failed.")
            raise dwmRequestError
      # Receive confirmation of request / error code
      handle_confirmation_TLV_frame()
      # Receive Node ID
      print("Handled confirmation frame.")
      [typeTLV, length, value] = receiveTLV()
      enforce_TLV_type(typeTLV, TLV_TYPE_NODE_ID)
      node_ID = parseTLV(typeTLV, length, value)
      return node_ID

# endregion


# region Helper Functions
```

```
def myParser():
      # This function handles command lets the user specify the
      # name of the port to use with a command line argument.
      # --port=[name or number]
      parser = argparse.ArgumentParser(description = 'get position info')  #
Describes what this script does
      parser.add_argument(
            '--port',
            default=defaultPortName,
            help='specify the name of the port to use (default: ' +
defaultPortName + ' )'
            )
      args = parser.parse_args()
      print("Using port:", args.port)
      return args.port


def sendTLV(request):
      """
      Sends a TLV request from the node to the Decawave unit.

      :param request: A TLV command, of type 'bytes'
      :return: EXIT_FAILURE if failed, otherwise EXIT_SUCCESS
      """
      global ser
      txBuffer = request
      try:
            ser.reset_input_buffer()   # Get rid of anything in the buffer
that could confuse this script.
            ser.write(txBuffer)
      except (serial.SerialException, AttributeError):
            print("Error during transmission of
request".format(txBuffer.hex()))
            return EXIT_FAILURE
      return EXIT_SUCCESS


def receiveTLV():
```

```
      '''
      Listen for TLV response from Decawave DWM1001 module.
      If it receives TLV_TYPE_DUMMY, it keeps listening for next message
      :return: The TLV response, as a list of 3 bytes objects.  [Type, Length,
Value]
      '''
      global ser  # The handle for the serial port connection
      typeTLV = TLV_TYPE_DUMMY
      while typeTLV == TLV_TYPE_DUMMY:
            typeTLV = ser.read(1)         #  Read  the  "type"  byte  of  the
response
      lengthTLV = ser.read(1)             #  Read  the  "length"  byte  of  the
response
      lengthTLV = int.from_bytes(lengthTLV, byteorder='little', signed=False)
      valueTLV = ser.read(lengthTLV)  # Read the value byte(s) of the response
      return [typeTLV, lengthTLV, valueTLV]


def parsePOSvalue(value):
      """
      This is a helper function to break a 13-byte position code into its
      individual components
      :param value: A 13-byte position code
      :return: the set of four integers [x, y, z, q] from the input code
      """

      x = int.from_bytes(value[0:4], byteorder='little', signed=True)
      y = int.from_bytes(value[4:8], byteorder='little', signed=True)
      z = int.from_bytes(value[8:12], byteorder='little', signed=True)
      q = int.from_bytes(value[12:13], byteorder='little', signed=True)
      return [x, y, z, q]


def parseTLV(typeTLV, length, value):
      # TLV_TYPE_DUMMY = bytes.fromhex("00")  # Reserved for SPI dummy byte
      #  TLV_TYPE_POS_XYZ  =  bytes.fromhex("41")    #  Response  position
coordinates x,y,z with q
```

```
      #  TLV_TYPE_CFG  =  bytes.fromhex("46")    #  Response:  configuration
information of DWM
      # TLV_TYPE_RNG_AN_DIST = bytes.fromhex("48")  # Response: Ranging anchor
distances
      # TLV_TYPE_RNG_AN_POS_DIST = bytes.fromhex("49")  # Response: Ranging
anchor distances and positions
      if typeTLV == TLV_TYPE_POS_XYZ:                        # 0x41 / Decimal
65
            [x, y, z, q] = parsePOSvalue(value)
            return [x, y, z, q]
      if typeTLV == TLV_TYPE_CFG:  # 0x46 / Decimal 70
            # Byte 1 (value[1]) parsing
            anchor = (value[1] & 0x20 != 0x00)  # Bit 5: TRUE if Anchor, FALSE
if Tag
            # (bit 4)   initiator
            # (bit 3)   bridge
            # (bit 2)   stnry_en
            # (bits 0-1)      meas_mode
            # Byte 0 (Value[0]) parsing
            # (bit 7)   low_power_en
            # (bit 6)   loc_engine_en
            # (bit 5)   enc_en
            # (bit 4)   led_en
            # (bit 3)   ble_en
            # (bit 2)   fw_update_en
            # (bits 0-1)      uwb_mode
      # Right now, I only care about tag/anchor status.  May change later.
            return anchor
      if typeTLV == TLV_TYPE_RNG_AN_DIST:             # 0x48 / Decimal 72
            # This code may be received from an anchor node after asking for
range information
            num_distances = int.from_bytes(value[0:1], byteorder = 'little',
signed=True)
            distances = []
            for i in range(num_distances):
                  offset = i*13+1
                  addr = value[offset:offset+8].hex()  # Note: Address size
is 8 bytes here, not 2 bytes
```

```
                  d = int.from_bytes(value[offset+8:offset+12], byteorder =
'little', signed=False)
                  dq = int.from_bytes(value[offset+12:offset+13], byteorder
= 'little', signed=False)
                  distances.append([addr, d, dq])
            return distances
      if typeTLV == TLV_TYPE_RNG_AN_POS_DIST:   # 0x49 / Decimal 73
            num_distances = int.from_bytes(value[0:1], byteorder = 'little',
signed=False)
            distances = []
            for i in range(num_distances):
                  offset = i*20+1
                  addr = value[offset:offset+2]   # UWB address in little
endian format
                  addr = addr[::-1].hex()   # Reverse order to big endian,
then make into string format
                  d = int.from_bytes(value[offset+2:offset+6], byteorder =
'little', signed=False)  # distance
                  dq = int.from_bytes(value[offset+6:offset+7], byteorder =
'little', signed=False)  # distance quality
                  [x, y, z, q] = parsePOSvalue(value[offset+7:offset+20])
                  distances.append([addr, d, dq, x, y, z, q])
            return distances
      if typeTLV == TLV_TYPE_NODE_ID:                 # 0x4E / Decimal 78
            nodeID = value[::-1]  # Reverse the bytes from little-endian style
to big-endian style
            #nodeID = value[:]
            return nodeID
      # Default case:
      print("Error: attempted to parse TLV of type not yet supported.")
      raise dwmUnexpectedMessage("Attempted to parse a TLV type not yet
supported")


def printTLV(typeTLV, length, value):
      if typeTLV == TLV_TYPE_POS_XYZ:
            [x, y, z, q] = parseTLV(typeTLV, length, value)
            print_POS_XYZ(x, y, z, q)
```

```
      if typeTLV == TLV_TYPE_RNG_AN_POS_DIST:
            distances = parseTLV(typeTLV, length, value)
            print_RNG_AN_POS_DIST(distances)
      if typeTLV == TLV_TYPE_RNG_AN_DIST:
            distances = parseTLV(typeTLV, length, value)
            print_RNG_AN_DIST(distances)


def print_POS_XYZ(x, y, z, q):
      print("{:_<15} {:_<15} {:_<15} {:_<5}".format('x', 'y', 'z', 'q'))
      print("{:<15} {:<15} {:<15} {:<5}".format(x, y, z, q))


def print_RNG_AN_POS_DIST(distances):
      print("{:=<5}     {:=<15}     {:=<5}     {:=<15}     {:=<15}     {:=<15}
{:=<5}".format('addr', 'd', 'dq', 'x', 'y', 'z', 'q'))
      for i in range(len(distances)):
                  [addr, d, dq, x, y, z, q] = distances[i]
                  print("{:<5}   {:<15}   {:<5}   {:<15}   {:<15}      {:<15}
{:<5}".format(addr, d, dq, x, y, z, q))


def print_RNG_AN_DIST(distances):
      print("{:=<5} {:=<15} {:=<5}".format('addr', 'd', 'dq'))
      for i in range(len(distances)):
            [addr, d, dq] = distances[i]
            print("{:<5} {:<15} {:<5}".format(addr, d, dq))


def handle_confirmation_TLV_frame():
      # Receive confirmation of request / error code.
      # Call after sending TLV request to inspect the first TLV frame.
      global ser
      [typeTLV, length, value] = receiveTLV()
      if value != ERR_CODE_OK:
            print("Received an error message.  Flushing input buffer.")
            print(typeTLV.hex())
            print(length)
```

```
            print(value)
            ser.reset_input_buffer()
            raise dwmRequestError


def enforce_TLV_type(received, expected ):
      if received != expected:
            caller_name = inspect.stack(1)[1][3]  # Get name of function that
called this function
            print("{0}  received  unexpected  TLV  value.    Flushing  input
buffer.".format(caller_name))
            print(received)
            ser.reset_input_buffer()
            raise dwmUnexpectedMessage
# endregion


# region Module Exceptions
class dwmError(Exception):
      # Base class for exceptions in this module
      pass


class dwmTransmitError(dwmError):
      # Used when transmission to dwm fails
      pass


class dwmRequestError(dwmError):
      # Use when a TLV request returns something besides 0x00
      pass


class dwmUnexpectedMessage(dwmError):
      # Use when getting a TLV message you did not expect to parse.
      pass
# endregion
```

```
# The following lines allow this script to run as a program if called directly.
if __name__ == "__main__":
      try:
            main()
      except rospy.ROSInterruptException:
            print("ROS interrupt Exception occurred")
            pass
```

Appendix D: Custom ROS Messages to Relay DWM Communications

Following are the message formats for the custom ROS messages that are being broadcast after a ROS node reads directly from a DWM via the TLV interface

```
#dwmPositionData.msg
Header header
string dwmID
bool isAnchor
bool positionIsValid
int32 x
int32 y
int32 z
char q
int32 numRanges
# dwmRange is defined a custom-defined message type
dwmRange[] ranges
```

```
#dwmRange.msg
string dwm_ID
int32 x
int32 y
int32 z
char position_quality
bool position_is_valid
uint32 distance
char distance_quality
```