Simultaneous Localization and Mapping

with Multi Robot Map Joining

John Downs

Feb 2013

Abstract

This paper presents an overview of Simultaneous Localization and Mapping. The focus is on getting the reader familiar with the essential concepts of the topic. Key ideas include the robot model, probabilistic interpretations of motion and observation, and map representation. There is a discussion of the Extended Kalman Filter with an aim towards getting the reader ready to make a simple implementation of SLAM. Finally there is a description of the state of the art in SLAM.

*** The focus of the Spring semester's work was the development of a SLAM implementation using submaps and map joining that was applicable to both single and multi robot scenarios. Map joining SLAM can reduce the dimensionality of the problem and simplifies the data association when sharing maps between robots. This paper covers some of the mathematical concepts such as least squares optimization and data association. Also discussed is the relationship between traditional least squares SLAM approaches and map joining.

Chapter 1

1

1.1 Introduction: What is SLAM?

A mobile robot in an unknown environment has two tasks that precede anything else. First, it needs a map of where it has been. The robot must ask 'what does my environment look like.' Second, it must localize itself using that map. The question becomes 'where am I?' In any partially observable environment, these two tasks are inseparable. The two tasks, localization and mapping, are combined into a single problem known as Simultaneous Localization and Mapping, or SLAM. Along with motion planning, these tasks for the basis for all mobile robot navigation. While not discussed in

detail here, localization, mapping and planning are applicable to manipulator arm robots as well as wheeled mobile robots.

Robot navigation may seem simple at first glance. Simply measure the location of objects in the environment and move between a pair of points. The problem with this is sensor noise. Every measurement has a limit to it's accuracy. This introduces error. All sensors can suffer from some systemic errors as well; wheels can slip, cross talk can affect sonar, and so on. The consequence is uncertainty in the position of environmental features and a robot's position. If left unchecked, that uncertainty will grow without bounds and invariably results in the catastrophic failure of any navigation scheme. Effectively dealing with this uncertainty requires some sort of probabilistic model.

Such a model is used to create an estimate of the map and the robot's position. As the robot continue to operate in the environment, it becomes more confident about the shape of that environment. The movement and observation are highly correlated because the observations depend on the position of the robot. Due to that correlation, the certainty about the robot's position also improves over time. The end result is an estimate of the map and a statement about its certainty.

Prior to a discussion about a probabilistic model, a discussion of deterministic robotics is necessary to provide a foundation. Two models will be developed that describe the essential behavior of the robot: the motion model and the observation model. These two models will give a framework for creating a probabilistic interpretation.

1.2 Motion

A robot's pose is a combination of the position and orientation, represented as a vector. In the two dimension case, this is represented by the column vector $[xy\theta]^T$ with position (x, y) and heading θ. The motion model describes the state transition a prior pose at time $t-1$ and a new pose at time t given a control input u. Several simplifying assumptions are often made. Most models use kinematics, a simple mathematical description of motion, rather than dynamics, which focuses on the forces responsible for motion. In it's simplest form, robot motion can use a particle like model consisting of linear and angular velocity in a plane. This extends to three dimensions without difficulty.

A derivation of the model begins with decomposing the components of

motion. There are three components: x position, y position and heading θ. The control signal u consists of a linear velocity $v\frac{m}{s}$ and angular velocity $\omega\frac{rad}{s}$. Consider linear velocity v with a heading θ, with an angular velocity of 0 from an initial position (x, y). Velocity is the first derivative of position r with respect to time, $v = \div dr, dt$. The the x' component is given by $\int 0, tx + vcos\theta dt$, the y' component is $\int_0^t x + vsin\theta\, dt$. The heading remains unchanged. If the angular velocity ω is non-zero and the linear velocity is zero, the change in position is also zero, but the change in heading is $\theta' = \int 0, t\theta + \omega dt$.

Combining the translational and angular velocities complicates the model. Assume that the robot travels with a constant velocity for a time interval. This is a realistic assumption for almost all scenarios where the interval is small enough. Rather than moving along a line or on a point, it now moves along an arc A with a radius r. The new radius is calculated by:

$$\vec{x}_{t+1} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{\omega}sin(\theta) + \frac{v}{\omega}sin(\theta + \omega\delta t) \\ \frac{v}{\omega}cos(\theta) - vcos(\theta + \omega\delta t) \\ \omega\delta t \end{pmatrix} \tag{1.1}$$

citeThrunPR See Appendix C for Proof.

Special consideration must be given to the case of motion in a straight

line. If the motion were strictly linear, the angular velocity will be 0, so the formula given in 1.1 would involve division by 0. In this case, the formula is instead:

$$\vec{x}_{t+1} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \begin{pmatrix} x + vsin\theta \\ y + vcos\theta \\ \omega + 0 \end{pmatrix} \tag{1.2}$$

Robotic motion can take on a variety of forms, from wheels to legs to flying rotors. The most common drive systems are either car-like or differential drives. Car like systems are very effective for outdoor experiments with uneven terrain, but are more difficult to model. Differential drives are better for indoor experiments and is remarkably simple to model, but has weaknesses that make it difficult to model in sloped environments. The differential drives consist of two wheels separated by an axle. Each wheel is controlled by a different motor and turns independently on the axle. This configuration allows the robot to turn in a circle either centered on a wheel or the axle.

The translation between the particle model and differential drive model is important. When we think of how something moves, we think of how fast it is moving in a direction and how fast it is turning. It also provides

a common format for controls that can be used independently of the wheel configuration. But there needs to be a correspondence between this idea and the actual motion of the wheels on the robot.

The motion of a differential drive comes from the velocity of its two wheels and their relative distance. Consider a differential drive robot traveling along an arc. Rotation is caused by a difference in speed between the right and left wheel that is proportional to the distance between those wheels l. Linear velocity is simply the average of the two wheel velocities.

$$\omega = \frac{v_r - v_l}{l} \tag{1.3}$$

citeDUDEK

$$v = \frac{v_r + v_l}{2} \tag{1.4}$$

1.3 Observation

The observation model describes how the robot's sensors are used to measure its environment. There are many kinds of sensors, but they can be divided first into two categories: those that measure an aspect of the robot

state and those that measure an aspect of the environment. We are concerned here with those that measure the environment. Some examples include sonar, laser range finders and cameras. For mapping, it is best to abstract the details of perception into range and bearing measurements. If the robot knows the relative distance and orientation of an object, it is relatively easy to put that into a global frame of reference. Features in the environment are usually represented as a set of points. Other representations, notably geometric and topological, are sometimes used, but will not be considered here CITESIEGWART.

A range sensor is usually fixed on a robot such that we know its orientation. When used, such a sensor will return the distance to the nearest object directly in front of it. The robot then has an estimated range and bearing of an object relative to itself. This is precisely the definition of a polar coordinate. Converting to Cartesian coordinates from a range r and bearing θ is done with:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} rcos\theta \\ rsin\theta \end{pmatrix} \tag{1.5}$$

We will also frequently need to use the inverse of this model as well.

This is just the conversion from Cartesian coordinates to polar form:

$$\begin{pmatrix} r \\ \theta \end{pmatrix} \begin{pmatrix} \sqrt{x^2 + y^2} \\ arctan(\frac{x}{y}) \end{pmatrix} \tag{1.6}$$

While not strictly a property of the sensor, a correspondence variable s is usually added to the tail of the observation vector. This is used to identify it as either a landmark that has been seen previously or a new landmark. If the landmarks are not easily identifiable, there are many classification algorithms that will match them. A characteristic algorithm, Maximum Likelihood, will be covered later.

1.3.1 probablistic models

A naive mapping algorithm would use the motion model to keep track of the robot's position, and the observation model to locate landmarks. The flaw in this approach is that no matter how accurate the sensors are, the error will be additive, and thus grow without bounds. An additional model is needed: a model of the uncertainty the robot has about its position and the position of environment features. The structure developed is known as a stochastic map CITEcheeseman1987stochastic. It consists of an estimate of the robot's pose and surrounding landmarks along with a covariance matrix

describing the confidence in that estimate. As a robot travels through an environment making observations, the correlation between landmarks increases, converging towards an accurate map.

Consider a pair of features, each some uncertain distance from the robot. The sensors will measure the distance as some value with a degree of error. The robot then moves to another position and again observes these two landmarks. This second observation reinforces the relationship between the landmarks, reducing the uncertainty of their relative distance. This reduction in uncertainty comes because the measurement error is inherent in the robot, so all the landmarks share the same error. As a network of relationships between landmarks and robot poses grows, the map tends towards an accurate relative representation CITEdurrant2006simultaneous. This increasing certainty in landmark position allows the robot to accurately determine its own position given an observation of a feature it has seen before.

Movement is described by:

$$p(x_t|x_{t-1}, u_t) \tag{1.7}$$

This gives the probability of the robot being at a particular pose x_t

given its last pose at time $t-1$ and a control input, u. A possible source of confusion is the use of x as the pose variable; x is a vector describing the position and heading angle, $[x, y]$. This is separate from the x of the Cartesian coordinates of the robot and should be clear from context.

Observation is described by:

$$p(z_t|x_t) \tag{1.8}$$

This model gives the probability that there is something at position z, that was correctly observed given the current pose of the robot, x, at some time t. Usually the robot will want to also keep track of landmarks it has seen, so it is common to add a map m.

$$p(z_t|x_t, m_t) \tag{1.9}$$

These probabilistic models allow us to use a predictor-updater style algorithm known as the Recursive Bayesian Estimation. Based on a prior state estimation, plus new information from observation and a control signal, a new state estimate is formed.

Intuitively, the Bayes Filter involves the robot moving to a position and guessing where it is. It then looks around and used that information to

Algorithm 1 Recursive Bayesian Estimation

Require: Prior estimate $prior(x_{t-1})$

Control signal u_t

Observation z_t

for $i = 0 : t$ **do**

$prediction(x_t) := \int p(x_t|u_t, x_{t-1}) prior(x_{t-1}) dx_{t-1}$ ▷ Prediction Step

$correction(x_t) := \eta p(z_t|x_t) prediction(x_t)$ ▷ Correction Step

end for

return $correction(x_t)$

improve the guess. More rigorously, the filter begins with an initial estimate of the state x_0, which is used as the base prior. The state is represented by a probability density function. The filter also requires a control signal u_t and observation z_t. The first step of the for loop predicts the new state of the environment based on the robot's motion. In a static environment, only the pose should change. This estimate includes a belief about the position of landmarks relative to the robot. It's inclusion in the next step tells us how much information is provided by the new observations, which is the probability that an observation z was made given the likely state x_t. The new information is used to create a better estimate of the state. This

is repeated until the system stops.

Because the two steps in the for loop of the Bayes Filter do not usually have a closed form, it's not very useful in practice CITEThrunPR. A number of algorithms approximate the Bayes filter. The most commonly used is the Extended Kalman Filter or EKF. The EKF is similar to the simpler Kalman Filter, with the addition of a linearization step that makes it applicable to non-linear systems.

1.3.2 EKF

In order to implement the EKF, we need to begin with data structures. First, we assume that the noise in the motion and observation models is Gaussian. While this is not necessarily true, it does allow for the relative simplicity of the EKF and a tractable estimate is often preferable to a more precise estimate. We will assume discrete time steps with a constant interval. The mean state estimate x is a column vector that combines the robot's pose at the current time t $R_t =< x_r, y_r, \theta_r >^T$ and the position of all the observed features $f_i \in F$, $f_i =< f_i x, f_i y >^T$. It's size is $1xN$, where N is the length of the pose and the combined length of the features. The covariance P describes the the confidence in each landmark's position and

has a size NxN.

1.3.3 Algorithms

Early SLAM research made heavy use of Bayes Filters such as the Extended Kalman Filter and Particle Filter to create a large global map of features in a robotâĂŹs environment. This approach worked well for small scale environments with a limited number of features, but proved to be inadequate for more expansive and feature rich environments. As the number of landmarks increases, computation time grows and error due to linearization have a greater effect. This was originally known as the online-SLAM problem, the idea being that a robot could use an online-SLAM algorithm to create a map in real time and use it for planning purposes.

The online SLAM problem is contrasted with the Full SLAM problem. In this case, the entire trajectory is retained, along with the relationship of the various poses to landmark observations. Naively, this formulation seems to be more complex and thus was not considered a viable solution early on for real time planning. However, the techniques developed turned out to be faster than filters in practice.

A drawback to most Full SLAM solutions is their high dimensional-

ity and the complexity of the implementations. Another approach, map joining, can reduce the dimensionality of the problem and the error due to linearization, while having simpler implementations. It is also easily adaptable to multi-robot scenarios. When two maps share a robot pose, either through being at sequential time steps or through observation of another robot, it becomes fairly easy to find the correct coordinate frame transformations to create a global map.

SLAM comes in two variants, online SLAM and full or batch SLAM. Online SLAM is used when the robot needs to maintain a map estimate at all times. Batch SLAM is used after data has been collected, but in practice proves to be fast enough to use in many real time scenarios.

The Extended Kalman Filter (EKF) is by far the most common state estimation algorithm used in SLAM. It is based on the Kalman Filter, but works non-linear functions, whereas the Kalman Filter works only for linear functions. The EKF begins by linearizing the odometry and observations, then applies the basic Kalman Filter algorithm. Like other Bayesian state estimators, the EKF is a two step algorithm with a time update and an observation update. The EKF is also in a class of filters known as Gaussian Filters. These filters make the assumption that the noise: that it is nor-

mally distributed. Several other variants of the EKF exist, most notably the Unscented Kalman Filter (UKF), discussed in CITEUKF.

The EKF relies on two other functions, commonly denoted as g and h with noise ϵ_t and δ_t:

$$g(u_t, x_t - 1) + \epsilon_t, \tag{1.10}$$

and

$$h(x_t) + \delta_t. \tag{1.11}$$

These are respectively the motion and observation models discussed earlier. These functions can be linear or non-linear, but in any case, they must be differentiable. EKFs use first order Taylor Expansion to linearize. This is simply the partial derivate of both g and h with respect to x_{t-1}. In order to ensure the linear approximation is accurate, the partial derivative is evaluated at μ_{t-1} for both g and h. These new functions are often referred to as the Jacobian matrices (or simply the Jacobians) of g and h, symbolized by ∇g and ∇h respectively. Note than when implementing an EKF, it is not necessary to calculate the derivative every time, but the value of the Jacobian will vary at each time step because it is a function. Several pseudocode implementations of the EKF can be found in CITEThrunPR2005.

One key limitation of the EKF is that the noise in odometry and sensors must be normally distributed, or Gaussian. In cases where the noise is likely to be normal, such as indoor environments with consistent surfaces, the EKF is probably a great choice, but in expansive outdoor environments, between the types of sensors and the properties of the terrain, other methods are preferable.

Another limitation associated with the EKF is that it invariably becomes inconsistent after long runs by becoming overconfident about the estimated state CITEJulier01acounter. In some cases, this causes a catastrophic estimation failure that is nevertheless difficult to detect. If great care is taken, it is possible to keep the growth of the inconsistency slow CITEBaileyNGSN06. But, while the inconsistency can be mitigated, it can never be eliminated. The consequence is that EKF SLAM is not suitable for long term exploration. Still, it is an excellent tool for short term, non-critical exploration tasks.

There are a number of alternative algorithms to the EKF. One popular choice is the particle filter. Particle filters belong to a class of non-parametric filters. This simply means that they do not explicitly model a particular probability distribution, but instead use Monte Carlo methods

to take a number of samples of possible states CITEThrun02d. The *particle* is a single sample, generated by taking the measured state and adding random process noise to the measurement.

An interesting property of particle filters is that their computational complexity can be modified to fit available resources CITEThrunPR2005. While this ability to reduce complexity comes at the cost of accuracy, it is often better to make some timely estimate rather than wait for something more precise. They have been used with great success in the popular FastSLAM algorithm CITEmontemerlo2003fastslam.

The EKF and Particle Filter algorithms are both online algorithms. Offline or batch algorithms are increasingly a subject of research. The key property of batch SLAM is the maintenance of the robot trajectory rather than just the current pose. An excellent example of this class of SLAM algorithms is GraphSLAM CITEThrun05GS. It possesses several properties used in advanced SLAM implementations: sparseness and the information form of the covariance matrix.

The *graph* in GraphSLAM refers to a graph of information constraints describing the robot's knowledge of the map and trajectory. The graph consists of two node types and two corresponding edge types. Nodes are

either robot poses or observed landmarks. Edges link either sequential poses or poses to landmarks. The constraints are expressed as edge weights calculated by

$$(z_t^i - h(z))^T Q_t^{-1} (z_t^i - h(z)) \tag{1.12}$$

and

$$(x_t - g(x))^T R_t^{-1} (z_t^i - g(x)), \tag{1.13}$$

where z_t^i is the i^{th} observation at time t, Q is the measurement covariance matrix and h is the measurement model in the first equation, and x_t is the robot pose at time t, R is the observation covariance matrix and g is the motion model in the second equation. The resulting graph is sparse, as opposed to fully or nearly fully connected. Because of this, updates are always local and do not suffer from the poor performance of full matrix updates. The graph can also be represented as an information matrix, which is essentially a weighted adjacency matrix. This corresponds to the covariance matrix used in traditional SLAM and is in fact just the inverse of that covariance matrix. There is also an associated information vector that corresponds to the usual state vector.

GraphSLAM produces two estimates: one for the robot trajectory and another for the map. An intuition of the graph is to consider vertices

and edges as masses attached to springs CITEThrun05GS. To estimate the trajectory, landmarks and their corresponding links to poses are removed and the eliminated links are added back into the edges connecting just the poses such that the *spring force* is maintained. The map is calculated by using the information matrix produced by the trajectory estimate step. The information matrix is inverted to give a covariance matrix and this is multiplied by the information vector to produce a state estimate.

1.4 Literature Review: State of the Art

Square Root Smoothing and Mapping ($\sqrt{SAM}$) is a full SLAM algorithm developed by Frank Dellaert and Michael Kaess that represents a break from earlier filtering approaches. Rather than filtering, it uses statistical smoothing methods over the robot trajectory. Smoothing over the entire trajectory turns out to be more efficient in practice that the EKF once the number of features exceeds 600 CITEDellaert06ijrr. Where filtering assumes that the current state is complete at each step when it is actually a linearized estimate, smoothing uses each pose in the trajectory to help calculate the most likely state.

Incremental Smoothing and Mapping (iSAM) is an extension of $\sqrt{SAM}$ that takes advantage of sparseness inherent in the full SLAM formulation to make only incremental updates as necessary CITEKaess08tro. The incremental nature of the algorithm means it can handle data as it comes in, which allows it to be run on an actual robot in service. iSAM does SLAM by formulating each step as a least squares problem. The result is the error in the map is a linear function that can be solved by the Gauss-Newton optimization method.

iSAM is often used as a map optimization tool for graph based SLAM applications CITESunderhauf. A mobile robot that is operating in an environment can add nodes and edges to a graph, just as in GraphSLAM. Once the map is needed for planning, iSAM can be run on the graph, even if the graph is extremely large, and return a result in a matter of seconds. While this is still not quite fast enough to make some time critical decisions, it is a great leap forward for SLAM algorithms.

Iterated Sparse Local Submap Joining (I-SLSJF) is yet another approach that tries to reduce the complexity of SLAM. In this case, the reduction comes from dividing the problem into a series of overlapping submaps CITEhuang2008iterated. Where graph based SLAM first esti-

mates a graph of poses, submap joining works more like traditional SLAM in that it focuses on the estimation of feature locations. As a side effect of the submap joining process, a coarse estimate of the robot trajectory is produced. But because the entire trajectory is not maintained, the number of dimensions in the problem is greatly reduced when compared to other batch SLAM solutions.

Submaps are usually produced with an EKF or other simple estimation technique and consist of a single step with a start and end pose in a local frame of reference. The end pose of a map is always the start pose of the next map in order to facilitate the fusing process. The initial version of the algorithm uses an information filter to match observations shared between maps. This filtering can still suffer from some linearization errors, so if an inconsistency is detected at any step, it can apply smoothing to the global map to recover from this inconsistency.

Another area of very active research is cooperative multi-robot mapping CITE(wang2007multi) CITE(multiSEIF), CITE(4399142), CITE(4543634), CITE(5509154). Outside of SLAM research, robot groups are popular because of improved redundancy and the obvious benefits of being able to execute a task in a distributed manner. Often multi-robot systems consist

of lower cost components because the group as a whole has a higher fault tolerance than any individual member. If an individual fails, the impact on the completion of the overall goal is mitigated. This can be useful in situations where the individual chance of failure is relatively high, or the cost of failure is high.

There are two challenges associated with multi-robot SLAM: distinguishing robots from landmarks and the determination of a shared frame of reference. Mistaking a robot for a landmark and adding it to the map can lead to an extremely inconsistent landmark. If a robot in the team is observed and classified as a landmark in one location and then observed again at another location, the observer may conclude that a loop has been closed, when this has in fact not occurred. The simplest way to cope with this is for the team to have a priori knowledge of the other members and devise some way to uniquely distinguish them from the environment, such as a barcode or unique pattern of infrared flashes. The other alternative is to make the SLAM implementation robust in a dynamic environment, but this comes with unique data association challenges.

Determination of a shared reference frame can be done by sharing maps when a mutual pose observation occurs between two or more robots. Once

a robot determines that it has observed another robot, it can communicate with the other team member and share its map and current pose estimate. In the map thus shared, there will be an estimate of the position of the team member. This point can then be used like the shared start end end poses with I-SLSJF SLAM. Once the location of the two robots is determined, it becomes possible to calculate the correct rotation and translation vectors to align the shared map with the robot's local map.

1.4.1 Least Squares Estimates

The key to the performance of full SLAM solutions is the least squares formulation. The objective is to minimize the Mahalanobis distance between the predicted state and combined prediction and observation. Mahalanobis distance is a measure of similarity for two data sets. Applied to SLAM, this estimate uses the entire robot trajectory and observations to date to calculate a total state estimate. Because it uses the entirety of the data available, it is called smoothing, to contrast it with filtering, which only uses the current estimate, rather than all historical data. Smoothing finds a single state estimate that best fits all the available data. The Least Squares formulation of SLAM is to minimize: $f(x_t-1,u)^T P f(x_t-1,u)+h(z)^T R h(z)$

TODO: Check this equation! Something - where f()is the motion model with covariance P, and his the observation model with covariance R, uis the set of control inputs and z is the set of observations. Many methods exist for solving linear least squares problems, but the motion and observation models are almost never linear in practice. When a linear least squares problem is at hand, there is always a closed form solution, thus it can be solved in a single step. However, because SLAM is non-linear, it requires iterative methods to solve. The reason for this is that no methods exist for solving these types of problems, so we must search for a solution. A number of machine learning algorithms exist to solve these problems, both deterministic and stochastic.

1.4.2 Map Joining

The map joining approach to SLAM relies on the creation of submaps: local maps focused only on a subset of a trajectory and the immediately related observations. The creation of local maps is usually accomplished by either Extended Kalman or Information Filters. Markov Chain Monte Carlo methods have also been suggested (CITE). While these methods prove to be inconsistent, this inconsistency is only significant in large sets of ob-

servations. By limiting their scope, linearization errors, inconsistency and complexity can be held in check. The earliest paper I have found on map joining is [TARDOS 2002]. This describes the fundamental operations of map joining: transformation from a common observation into a global coordinate frame and feature association. More recent formulations [C-SAM] have eliminated the need for explicit transformation through the use of a graph theoretic formulation of SLAM. In this case, the more general term âĂŸmap alignmentâĂŹ is used over transformation. No matter how the maps are aligned, the second step is data association. Because of possible noise in the observation of a common landmark, alignment may not place all landmarks at the same point. This requires a classification algorithm to be run over the map, such as k-Nearest Neighbor or Joint Compatibility Branch and Bound [TARDOS 2002]. Other classifiers can be used, but are not common in the literature. Classification can be eliminated if noiseless identification of features is possible, such as when using cameras and unambiguous barcodes. If there are common landmarks between the two submaps, after classification, a new state estimate is required to make sense of the matched but non-coincident features. This is accomplished by calculating a least squares estimate of the new global map, to create a best

fit âĂŸcurveâĂŹ describing all the observations.

1.4.3 Spherical Matrices, One Step SLAM and Map Joining

A key observation to the reduction of dimensionality in single step SLAM is the use of spherical covariance matrices. An spherical matrix is defined as any matrix that is commutative with a rotation matrix. A rotation matrix R(theta) is [cos t -sin t; sin t cos t] âĘŘ CHECK THIS . For all theta and any spherical matrix A, AR = RA. This property allows for the reformulation of the Full SLAM function into Formula here It is important that the covariance matrices are also positive definite. A matrix is positive definite if for all positive, non-zero column vectors $v, v^T M v > 0$. This is to allow for methods analogous to finding the square-root of a matrix, such as Cholesky or QR factorization to be used in the solution of the least squares estimate. In the Automatica preprint, Dr. Huang claims that this objective function, when applied to single step SLAM, is equivalent to a one dimensional optimization problem. Through repeated application of single step SLAM for each timestep, an approximate solution to the previous objective function can be easily found. It is approximate because a

spherical covariance matrix is used, rather than the actual covariance associated with observation uncertainty. In On The Num of Local Minima, Dr. Huang shows that single step SLAM and map joining SLAM share the same property of having only 1 or 2 minima, one of which is global, when covariances are approximated by spherical matrices. This provides the benefits of the reduction in linearization error due to map joining, along with the low dimensionality of single step SLAM. Additionally, if multi-robot map joining belongs to this class of problems, map sharing can possibly become quite efficient. In a remark from On The Number.. H notes that the covariance matrices must only be spherical, but not identical. That is, the covariance can differ for each odometry measurement and observation. This lends some home to the possibility that multi-robot map joining is in this class of problems.

1.4.4 SLAM and Machine Learning

There are two sub-problems within SLAM that call for the application of machine learning techniques, landmark association and least squares optimization. These are two separate types of problems, the former being unsupervised classification, the latter is convex optimization. Landmark

association, also known as data association, is a classification problem that uses a pair of feature maps or a map and set of observations and tries to match known landmarks with new observations. In most cases, this is an unsupervised learning problem because the set of features can vary so greatly from map to map, it is not possible to provide examples for a supervised approach. This is necessary for any environment where there can be ambiguity in landmarks. While it might be possible in a lab to put barcodes on landmarks that can be recognized with a camera, in scenarios where a camera might not be available or barcoding landmarks unfeasible, landmark association is required. The least squares portion of SLAM is non-linear and of a very high dimension. Because of this, single step techniques such as linear regression are not applicable. Other algorithms such as Levenberg-Marquardt, Gauss-Newton and Stochastic Gradient Descent must be used instead. These three methods are the most popular among SLAM researchers. The Gauss-Newton method is used effectively in iSAM (CITE). It is a variation on NewtonâĂŹs method for finding the minima of a function taught in elementary calculus courses but is modified to solve least squares problems. It begins with an initial guess x_0 of a possible solution. For an objective function $f(x)$ and a residual function $r(x)$, the

jacobian at $r(x_0)$ is calculated. The jacobian is a linearization estimate of $r(x)$ and is used for the next guess. This process is repeated until it converges on a solution. It is possible in certain situations to overshoot the optimum and fail to converge. This algorithm can also fail to find a global optimum and instead converge on a local optimum. Gradient descent is an optimization algorithm similar to hill climbing, but rather than selecting a single element to improve, it follows the slope (or gradient) of the function towards a solution. Like Gauss-Newton, gradient descent can get stuck in local optima. A modification known as stochastic gradient descent can overcome this limitation.

1.5 Methods

The first attempt at implementing SLAM involved adapting the algorithm in [1] from the e-Puck to the Khepera III models for Webots. I designed a rectangular arena with several cubes for landmarks. The implementation was altered to move around the arena using a Braitenberg vehicle navigation strategy and to use sonar and infrared sensors rather than a camera for landmark detection. A total of three robots were added to the Webots

world. This simulation was plagued with numerical instability. Whenever a robotâĂŹs wheel would move backwards, this would cause the robot position to become undefined and would result in the model disappearing from the simulation. This was due to a bug in the code provided by [1]. After trying to resolve the issue with poor results, I decided to try a different approach. I also worked briefly with the Mobile Robot Programming Toolbox [2], attempting to work around the simulation instability, but found the documentation on the required features was too incomplete, the author having focused on Kinect functionality instead.

A new review of robotics toolkits provided by other researchers resulted in two findings. First, most implementations were in Matlab [3] [4], and those that werenâĂŹt tended to be older and poorly documented and unmaintained. Matlab seems to be used most often because of itâĂŹs high level mathematical libraries and ease of use (as it is an interpreted language). Second, most SLAM implementations used data files with velocity measurements and range/bearing measurements of landmarks. This is done to abstract away the particulars of a robot and instead focus on the details of SLAM. I then found an excellent multi robot data set that was in the correct form for doing SLAM [8]. At this point, I was able to move forward

again. The map joining approach is a continuation of the work done in [5], [6], and [7]. It first requires a sequence of submaps that share end and start poses at the transition from one to the next. These maps can be produced by any SLAM algorithm. I chose the EKF because it is the simplest to implement. The initial pass was buggy and lead to very poor maps, which later affected the quality of the global map state. This EKF implementation was later replaced by working through the tutorial in [9]. Each map consisted of two robot poses, the last pose from the previous map (except the first map, which has the initial pose), and the final pose. The first map was considered the âĂŸglobal mapâĂŹ that all subsequent maps were added to. The maps were combined by translating the new local map to the coordinate frame of the global map and then solving the least squares SLAM formulation provided in [6]. This least squares approach searches for a state vector that minimizes the square of observation error for each landmark. This equation was translated into a fitness function that could be used by machine learning algorithms to find a solution. In this case, I used a genetic algorithm using the combined local and global state vector as the gene string. Preliminary runs would converge to a solution in less than 100 generations, but it is not clear whether this behavior will continue

when used with more accurate maps.

The new one step EKF SLAM implementation needs to be combined with the map joining algorithm. This should just be a matter of refining the interface between these two components. The map joining approach will be compared to the EKF-only maps. The map joining algorithm needs to be made more modular so other machine learning algorithms can be applied. Finally, data needs to be collected from the Khepera robots to see if this approach can work with unknown data association. For multiple robots, I need to develop a way to detect when map sharing is appropriate. When a robot detects another one, there will be a common pose that can allow for the correct translation of the frames of reference and the same fitness function can be applied. It is still an open question as to whether this has the same number of local minima as the situations described in [5]. Grade: While I feel I did not make all the progress I wanted this Spring, I did complete a number of the pieces necessary to answer these questions. I have also come to understand a number of concepts involved with SLAM that were beyond the scope of the first part of this project. It was fair, but not outstanding work, so I feel a B would be appropriate.

1.6 Results

1.7 Conclusions

1.8 Discussions

1.9 Future Work

1.10 Appendix A

MATLAB Source Code

1.11 Appendix B

1.12 Appendix C

Various proofs $r = |\frac{v}{w}|$ https://www.khanacademy.org/science/physics/torque-angular-momentum/torque-tutorial/v/relationship-between-angular-velocity-and-speed : Uses dimensional analysis

1.13 Bibliography