1 Project Description

- Team: Anthony Dierssen-Morice
- Project Option: Localization for Navigation; Bonus #1: no use of landmarks' global location.

2 Robot Control

To control robot movement, a differential drive system was adopted. Provided with two control signals, a translational velocity V, and an angular velocity ω, the appropriate rotational velocity of each robot wheel–ω_{left} and ω_{right}–are determined according to the following equations:

$$\omega_{right} = \frac{V + \omega\frac{\ell}{2}}{r} \tag{1}$$

$$\omega_{left} = \frac{V - \omega\frac{\ell}{2}}{r} \tag{2}$$

where ℓ corresponds to the length of the axle running between robot's two wheels and r corresponds to the radius of the robot's wheels. Note, that if the division by r is removed from equations 1 and 2, we are left with the translational velocities for each of the wheels. Given that the Webot motor control instruction `some_motor.setVelocity(velocity)` takes as its argument a rotational velocity setpoint, however, it is necessary to include this division step so as to obtain the rotational velocities of each wheel for control.

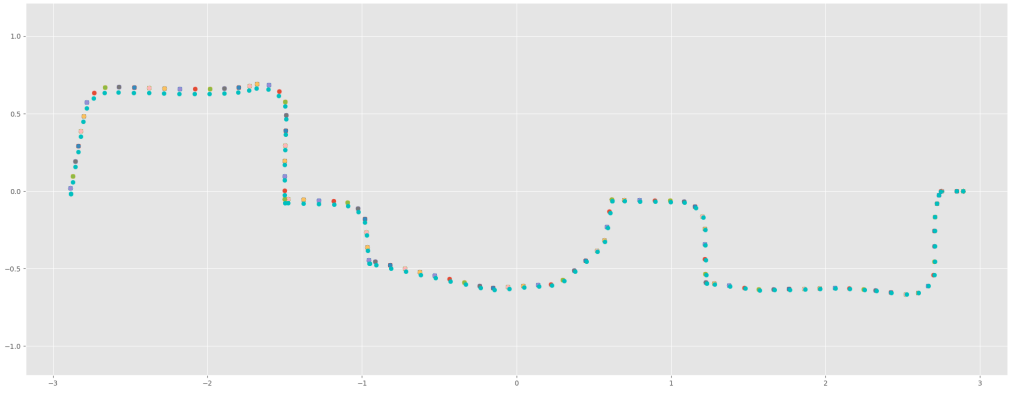
It is worth noting that V and ω can, themselves, be expressed in terms of the individual wheel velocities:

$$V = r * \left(\frac{\omega_{left} + \omega_{right}}{2}\right) \tag{3}$$

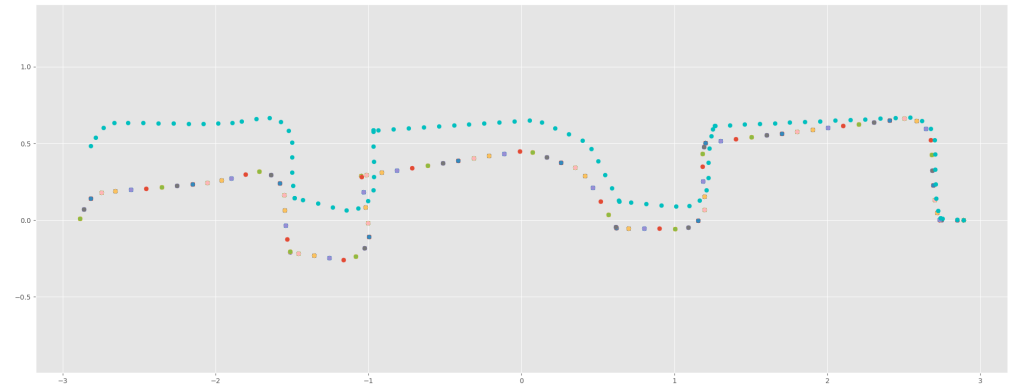
$$\omega = r * \left(\frac{\omega_{right} - \omega_{left}}{\ell}\right) \tag{4}$$

Critically, the values of r and ℓ must be correct for the system to behave as expected. While the Webots documentation lists the E-puck model parameters as $r = 20.5mm$ and $\ell = 52mm$, these values are not accurate. In actuality, the true values for wheel radius and axle length are $20.0mm$ and $57mm$ respectively. These true values can be found by inspecting the `boundingObject` of either of the wheels and observing that it is defined by a cylinder of radius $0.02m$ and height $0.005m$. The height of this `boundingObject` is important in its own right as even though the distance between the centers of the two wheels' `boundingObjects` is in fact $52mm$, it appears as though the contact point of the wheels is actually closer to the outer edge of the `boundingObjects`. This gives an effective axle length of approximately $0.052 + (0.005/2) * 2 = 0.057m = 57mm$. These discrepancies, particularly the difference in wheel radius, turned out to be the largest challenge encountered in the development of this project. Significant effort went into debugging poor localization performance. The debugging of this issue was further complicated by the fact that added noise in the control signals was actually masking the problem. It was only through experimentation with no control signal noise that the issue became apparent, since the robot's estimated orientation $\hat{\theta}$ after spinning through an angle of only 2π was found to be off by as much as 0.15 rad (roughly 8.6 degrees). After correcting these values, the same spin test resulted in virtually zero error in $\hat{\theta}$ when no control noise was added. The degree to which this error affects state propagation is shown in figure 1.

To provide movement which is as smooth as possible, despite the control input noise, a linear ramping scheme was implemented to smooth the transition between different control values. For example, if the path planning algorithm (described in subsequent sections) requests a translational velocity of $V_{trans} = 0.1m/s$, and the robot's current translational velocity is zero, the controller will linearly ramp from 0 to 0.1 over multiple time steps. The rate at which the control input will ramp is determined by constants `ACCEL_RATE_TRANS` and `ACCEL_RATE_ANGULAR`, for the translational and angular velocities respectively.



(a) $r = 0.020\ m$



(b) $r = 0.0205\ m$

Figure 1: Plot of estimated robot position (multicolored trail) and ground truth position (cyan trail) collected with zero control input noise during navigation to goal (moving left across plot), using different values for robot wheel radius r, and no EKF measurement update step.

3 Path Planning

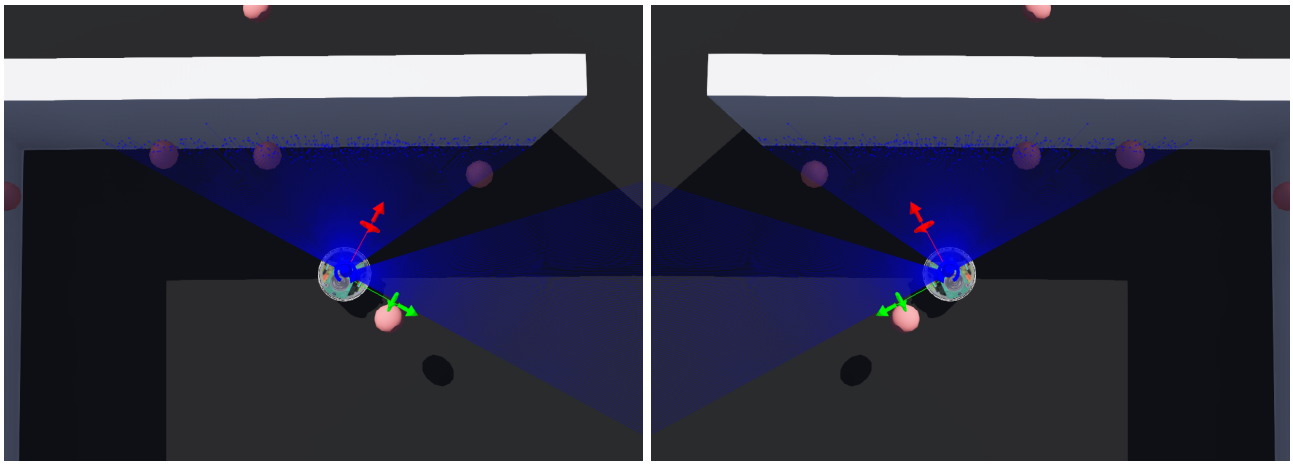
A local path planning algorithm was used to navigate from the known start position to the provided goal position. In particular, a hybrid left-right turning Bug 0 algorithm was implemented. In brief, this algorithm alternates between two states: `'go to goal'` and `'follow obstacle'`. Observe the following psuedocode:

1. (`go to goal`) Move in the direction of the goal until an obstacle obstructs progress.
2. (`follow obstacle`) Follow the obstacle in a specific direction until the path towards the goal is clear again.
3. Repeat steps 1 and 2 until the robot has reached the goal.

It is typical that a Bug 0 implementation is classified as either left-turning or right-turning to indicate whether the robot follows obstacles in a counter clockwise or clockwise direction respectively. In the provided test environment, however, such an implementation would result in lengthy travel times from the start position to the goal on account of the narrow entrances/exits from the square corridors causing lots of backtracking. As such, a hybrid left-right turning Bug 0 algorithm has been implemented which decides on the direction of travel around an encountered obstacle each time the algorithm transitions from the `go to goal` state to the `follow obstacle` state.

To determine the best direction of travel around an obstacle, the robot's lidar sensor is leveraged to provide a notion of the angle of approach to that obstacle. Specifically, the average distance readings of the first half $[\frac{\pi}{2}, 0)$ of the lidar

sweep are compared to that of the second half of the lidar sweep $(0, -\frac{\pi}{2}]$, noting that the total FOV of the lidar is $\theta \in [\frac{\pi}{2}, -\frac{\pi}{2}]$. Whichever half of the lidar readings has the larger average (with infinite readings clipped to the lidar's maximum range), is assumed to represent the side of the robot which lies interior to the oblique angle of incidence with the obstacle. It follows that if the right half of the lidar's readings have a greater average value than the left half, such a configuration would fall into the same category as that depicted in figure 2a. Conversely, if the right half of the lidar's readings have a smaller average value than the left half, such a configuration would fall into the same category as that depicted in figure 2b. This decision process leads to the robot following encountered obstacles in the direction that poses the least initial "resistance" (i.e. initially requiring the smallest change in direction).



(a) "Left-turning" (b) "Right-turning"

Figure 2: Example configurations that will result in either a 2a) left-turning, counter clockwise obstacle exploration, or a 2b) right-turning, clockwise obstacle exploration.

Note that this angle of incidence assumption is not guaranteed to hold in all environments, and is subject to lidar noise when the angle of incidence is close to perpendicular. In the provided environment, however, this simple implementation detail leads to a near optimal path from the start position to goal position.

4 Localization

An Extended Kalman Filter (EKF) based SLAM approach was implemented to fulfill the bonus #1 problem (i.e. no use of any landmarks' global position). Initially, no information is known about any of the landmarks. As a landmark is encountered for the first time, however, the measured relative position of the landmark to the robot is used to initialize its uncertainty stored in the covariance matrix Σ_x^t, and position stored in the state vector $\hat{x}^t$. The state vector contains the robots current mean estimate of its pose (x, y, and θ) along with the (un)initialized x, y coordinates of each of the landmarks at timestep t. This gives $\hat{x}^t$ a dimension of `(3 + 2 * NUM_LANDMARKS)` by `1`. Note that landmark correspondence is assumed to be given (i.e. landmark recognition object id's are used to determine which landmark corresponds to a specific measurement).

In the first step of the EKF, the state and uncertainty in the robot's state is propagated forward some time unit (dt)–given both the robot's current state and its control signals (V^t, ω^t)–using the following forward kinematic equations (note that the superscript $t+1|t$ means: ...at time t+1 given measurments up to time t):

$$x^{t+1|t} = x^t + V^t \cos(\theta^t) dt \tag{5}$$

$$y^{t+1|t} = y^t + V^t \sin(\theta^t) dt \tag{6}$$

$$\theta^{t+1|t} = \theta^t + \omega^t dt \tag{7}$$

Note that this state propagation step only changes the robot's pose and not the positions of the landmarks. The state covariance matrix, meanwhile, is updated according to the following similarity transformation:

$$\Sigma_x^{t+1|t} = G\Sigma_x^t G^T \tag{8}$$

where G is the jacobian matrix of the state update performed in the propagation step described above. As such, G is an identiy matrix except for the first `(3 x 3)` block in the top left which reflects the jacobian of the state update with respect to the robot's pose (x, y, and θ):

$$J_{x,y,\theta} = \begin{bmatrix} 1 & 0 & -V^t \sin(\theta^t)\ dt \\ 0 & 1 & V^t \cos(\theta^t)\ dt \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

An additional similarity transformation is then added to $\Sigma_x^{t+1|t}$, before returning from the first step of the EKF, so as to incorporate the control noise into the state unceratainty. Since the control noise does not impact the static landmark positions, this addition to $\Sigma_x^{t+1|t}$ will only change to the entries corresponding to the covariances of the robot's pose:

$$\Sigma_x^{t+1|t}[:3,:3] \texttt{ += } J_{V^t,\omega^t}\Sigma_u J_{V^t,\omega^t}^T \tag{10}$$

where J_{V^t,ω^t} is the jacobian of the robot's state with respect to the control input, and Σ_u is the covariance matrix of the control input:

$$J_{V^t,\omega^t} = \begin{bmatrix} \cos(\theta^t)\ dt & 0 \\ \sin(\theta^t)\ dt & 0 \\ 0 & dt \end{bmatrix} \tag{11}$$

The end effect of the EKF propagation step, then, is limited to the first 3 entries in the state vector along with the first three rows columns of the state covariance matrix.

In the second step of the EKF, the state vector and state covariance matrix are updated based on relative position measurements to visible landmarks. Note that this step is only performed every `UPDATE_FREQ = 200` timesteps, whereas the propagation step described above is performed at each timestep. Continuing, for each relative position measurement to a visible landmark L_i, the residual r_i between the measurement z_i and the predicted measurement $\hat{z}_i$ given the current belief of where that landmark's global position lies $(\hat{L}_{ix}, \hat{L}_{iy})$and the robot's current pose is calculated:

$$\hat{z}_i = \begin{bmatrix} \cos\hat{\theta}^{t+1|t} & -\sin\hat{\theta}^{t+1|t} \\ \sin\hat{\theta}^{t+1|t} & \cos\hat{\theta}^{t+1|t} \end{bmatrix}^T \left(\begin{bmatrix} \hat{L}_{ix}^{t+1|t} \\ \hat{L}_{iy}^{t+1|t} \end{bmatrix} - \begin{bmatrix} \hat{x}^{t+1|t} \\ \hat{y}^{t+1|t} \end{bmatrix} \right) \tag{12}$$

$$r_i = z_i - \hat{z}_i \tag{13}$$

With the measurement residual in hand, we then aim to calculate the Kalman gain K so that we might update the state vector according to the following equation:

$$\hat{x}^{t+1|t+1} = \hat{x}^{t+1|t} + K\ r_i \tag{14}$$

The Kalman gain is found using:

$$K = \Sigma_x^{t+1|t}\ H^t\ (H\ \Sigma_x^{t+1|t}\ H^T + \Sigma_m)^{-1} \tag{15}$$

Where Σ_m is the covariance matrix for the relative position measurements and H is the jacobian of the state vector with respect to the measurement residual. Note that this matrix will be sparse as only those columns corresponding to the robot's current pose, and the position of the landmark under consideration will have non-zero values:

$$H = \begin{bmatrix} \left(-\cos(\hat{\theta}^{t+1|t})\right) & \left(-\sin(\hat{\theta}^{t+1|t})\right) & \left(-\sin(\hat{\theta}^{t+1|t})(\hat{L}_{ix}^{t+1|t} - \hat{x}^{t+1|t}) + \cos(\hat{\theta}^{t+1|t})(\hat{L}_{iy}^{t+1|t} - \hat{y}^{t+1|t})\right) & 0 & \dots & \left(\cos(\hat{\theta}^{t+1|t})\right) & \left(\sin(\hat{\theta}^{t+1|t})\right) & 0 & \dots \\ \left(\sin(\hat{\theta}^{t+1|t})\right) & \left(-\cos(\hat{\theta}^{t+1|t})\right) & \left(-\cos(\hat{\theta}^{t+1|t})(\hat{L}_{ix}^{t+1|t} - \hat{x}^{t+1|t}) - \sin(\hat{\theta}^{t+1|t})(\hat{L}_{iy}^{t+1|t} - \hat{y}^{t+1|t})\right) & 0 & \dots & \left(-\sin(\hat{\theta}^{t+1|t})\right) & \left(\cos(\hat{\theta}^{t+1|t})\right) & 0 & \dots \end{bmatrix} \tag{16}$$

Finally, the state covariance matrix is updated as follows:

$$\Sigma_x^{t+1|t+1} = \Sigma_x^{t+1|t} - H^T\ (H\ \Sigma_x^{t+1|t}\ H^T + \Sigma_m)^{-1}\ H\ \Sigma_x^{t+1|t} \tag{17}$$

Note that in addition to the aforementioned use of landmark id's, it is further assumed that the number of landmarks which exist in the world is known. This assumption is made only to avoid dynamic resizing of matrices as new landmarks are encountered.

Shown in figure 4 is a plot of the robot's estimated position throughout its navigation from its starting position to the goal position using the above EKF-based SLAM method.

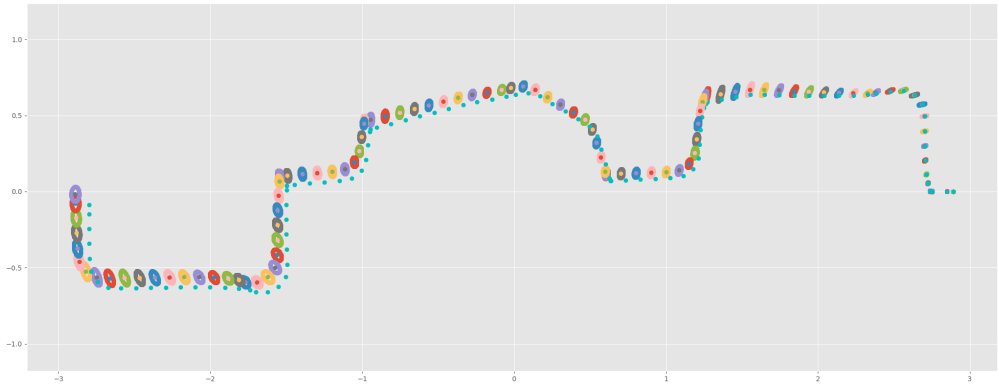


Figure 3: Plot of estimated robot position (multicolored trail) and ground truth position (cyan trail) collected during navigation to goal (moving left across plot) using EKF-based SLAM method.

5 Additional Difficulties

A problem frequently encountered in running the Webots simulation was finding the robot stuck after its wheels had clipped through the world's ground plane:

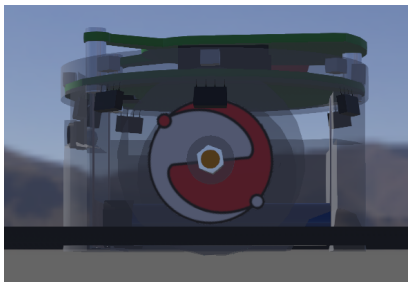


Figure 4: Robot stuck after clipping through the world's ground plane

To reduce the frequency of this problem, the WorldInfo parameters `CFM` (Contsraint Force Mixing), `ERP` (Error Reduction Parameter), and `basicTimeStep` were set to $1e-9$, 0.9, and 8 respectively to increase both the simulation precision and error correction. This fix, while effective in reducing the frequency of occurrence, does not entirely eliminate the issue. This means that the world must occasionally be refreshed with the default WorldInfo parameters before reimplementing the fix.

On the topic of world configuration, it is worth noting that the 200 landmarks spread across the world were instantiated using a helper script `world_builder.py` that has been including in the submitted source files. This script creates, and randomly distributes landmarks around the world, all at the same height. After their initialization, a handful of landmarks were moved manually with the intention of minimizing the time spent by the robot without seeing any landmarks.