Team 5 ROB 550 Balancebot Report

Sheng Zhong, Chengxi Feng, Poorani Ravindhiran

***Abstract*—This report describes the construction, control using sensor feedback and the navigation of a self-balancing robot. The performance of the balance bot is analyzed by a competition involving four tasks. The tasks check the effectiveness of the controller and the accuracy of the sensor data by making the robot manually as well as autonomously drive through different arenas. We show that we successfully constructed the robot; estimated model and motor parameters; implemented a classical controller using successive loop closure, a turning controller to change heading of robot and a controller to manually control the robot using a remote. We also developed odometric dead-reckoning functionality on the robot and implemented navigation algorithms.**

I. Introduction

IN this report, we will describe and discuss the implementation of a two-wheel self-balance robot using cascade Proportional-Integral-Derivative (PID) control and path-planning function. Section II shows the detailed design and implementation. The controllers take feedback from and performs odometry on the 48 CPR encoder on DC motor and MPU9250 Inertial Measurement Unit (IMU) which can measure the linear acceleration and angular velocity along 3 axes. Section III discusses the results and validation of our methods. We plot the step responses for each controller and discuss their sensitivity. We validate the odometry model, characterizing its accuracy. Finally, we evaluate the path-planning function using the data from the Optitrack motion capture system. The objective of this project was to build a Balancebot to perform in a competition consisting of four tasks. The tasks are defined as follows:

1) 4 Left Turns: The robot has to autonomously drive in a 1m square while always balancing using odometry.
2) Straight Line Drag Racing: The robot has to race down the hallway (11 m long) autonomously as fast as possible using odometry without falling over.
3) Manual Race: Robot should race through a given course 3 times while being driven manually, without falling over or hitting an obstacle. The given course consisted of four closed gates, which should be opened in only one direction.
4) Autonomous Race: The robot has to race through the same course, used in task 3, for 3 times under autonomous control.

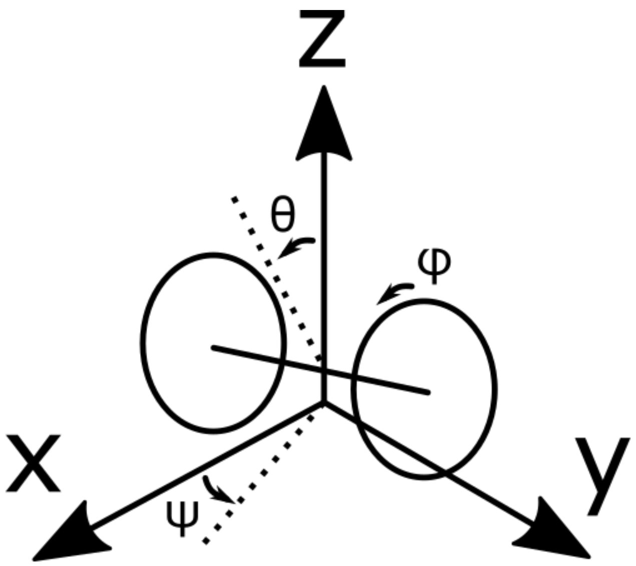


Fig. 1: Model of the Balancebot with relevant angles labelled. The axes represent the initial robot frame.

II. Methodology

The Balancebot is an unstable system requiring fast dynamic control to stabilize. Ignoring unequal wheel effects, it is equal to an inverted pendulum [1]. Its model is shown in Figure 1. We use heading and ψ interchangeably.

A. Motor Control

We designed a cascade controller illustrated by Figure 2. From left to right, the controllers' become lower level and have correspondingly smaller rise times to deal with faster dynamics. The parameters of each controller are in Table I. We restrict PWM supplied to motors from being within the range $[-0.05, 0.05]$ by setting it to the limit it's closer to, unless it's magnitude is $< 10^{-5}$. This is to overcome viscous friction.

Controller coefficients	K_p	K_i	K_d	t_{rise} (s)
Balance controller	8	10	0.22	0.0278
Heading Controller	0.4	0	0	0.6*
Position controller	0.0063	0.00077	0.0064	0.2

TABLE I: Low level controller parameters. t_{rise} is the desired rise time of a discrete PID controller marched at 100Hz. *The rise time of the heading controller was not explicitly designed for and is instead the experimental rise time seen in Figure 6.

Model Parameters Of Balancebot

The model parameters read from the Computer Aided Design (CAD) model of the Balancebot are listed in the following table. We constructed the Balancebot without modification to the CAD design.

Parameter determined	Value
Radius of wheel	0.04 m
Mass of one wheel	0.03 kg
Mass of the robot without wheels	0.802 kg
Center of Mass from wheel axis	0.1122 m
Moment of Inertia of center of mass about x-axis	5770 kg mm^2
Moment of Inertia of center of mass about y-axis	4492 kg mm^2
Moment of Inertia of center of mass about z-axis	3097 kg mm^2

TABLE II: Parameters determined from the CAD model of the Balancebot.

Determination Of Motor Parameters

The motor parameters required for modelling include no load speed (ω_{nl}), motor constant (K), stall torque (τ_s), viscous friction coefficient (b), motor coil resistance (R), and inertia of the motor shaft and gearbox (J). The resistance for each motor is measured across the coil ends using a multimeter. To determine the other parameters, we supplied $11.67 \pm 0.01V$ to each motor with a DC power supply. Analyzing the circuit model for the motor using Kirchhoff's Voltage Law gives the equation:

$$V = Ri + L\frac{di}{dt} + K\omega \tag{1}$$

In steady state, $\frac{di}{dt} = 0$. Hence,

$$V = Ri + K\omega \tag{2}$$

Applied voltage is measured with the multimeter, and the current is read from the motor driver's current sense pin. We determine angular velocity (ω) of the motor by dividing accumulated encoder ticks by the elapsed time. This gives average ω which is valid here because we start data collection only after we enter steady state (after 2 seconds).

The motor constant is then estimated with least squares with around 1000 points of data ($\omega^\dagger$ is ω's pseudoinverse).

$$K = \omega^\dagger (V - Ri) \tag{3}$$

From the torque balance equation, the developed torque (τ) can be expressed as a function of applied voltage

$$\tau = \frac{KV}{R} - \frac{K^2\omega}{R} - b\omega \tag{4}$$

The static friction coefficient (c) is ignored while modelling for convenience. When the motor is stalled, $\omega = 0$. Thus $\tau_s = \frac{KV}{R}$, where τ_s is the stall torque. When there is no torque developed, $\tau = 0$ and hence, no load speed is given as $\omega_{nl} = \frac{KV}{K^2+Rb}$.The friction coefficient can be determined from the above equation as

$$b = \frac{KV - \omega_{nl}K^2}{\omega_{nl}R} \tag{5}$$

We determined b by using the least squares estimate of K and average ω_{nl}, and estimate its uncertainty with the minimax method - by calculating its range of values when using input values at their uncertainty limits.

J is determined by first driving the motor to no load speed, then removing current and capturing its speed as it decelerates. We remove current because the current sense measure is noisy and setting duty cycle to 0 gives the least uncertainty in current supplied. $0 = J\frac{d\omega}{dt} + b\omega$. Unlike before, in this experiment we determine the instantaneous speed by differentiating once and the acceleration by differentiating twice. We then estimate J via least squares with around 40 points of data.

$$J = \dot{\omega}^\dagger (b\omega)$$

B. Balance Control

The balance controller is the innermost and fastest controller stabilizes θ such that $\theta_{set} - \theta \to 0$. θ_{set} is the output of the ϕ or position controller. The balance controller outputs a shared motor PWM, saturating at -1 and 1. We first model how θ responds to torque with its transfer function, developed in [2].

$$\begin{aligned}
a &= m_r R_w L + (m_r + mwR_w^2) \\
b &= -(I_w + (m_r + mwR_w^2)(I_r + m_r L^2)s^2 \\
c &= m_r g L(I + m_r + mwR_w^2) \\
G_1 &= \frac{\Theta(S)}{T(S)} = \frac{a}{b+c}
\end{aligned}$$

Another controller which is parallel to the balance controller is heading controller. It stabilizes ψ such that $\psi_{set} - \psi \to 0$. It outputs a PWM difference (add to right and subtract from left) to control a turn in place behaviour. The heading controller only has a proportional term because we did not notice any steady state error nor any oscillation with just a proportional term.

The position controller stabilizes ϕ such that $\phi_{set} - \phi \to 0$. It outputs a θ_{set} to the balance controller, and does not control PWMs directly. Transfer function 6 shows how the wheel angle ϕ changes with θ [2].

$$G_2 = \frac{\Phi(S)}{\Theta(S)} = \frac{-(I_r + m_r R_w L)s^2 + m_r g L}{(I_w + (m_r + mwR^2 + m_r R_w L)s^2} \tag{6}$$

By including the motor dynamics directly in the transfer function, we are able to design the balance controller and position controller using equation 7 8

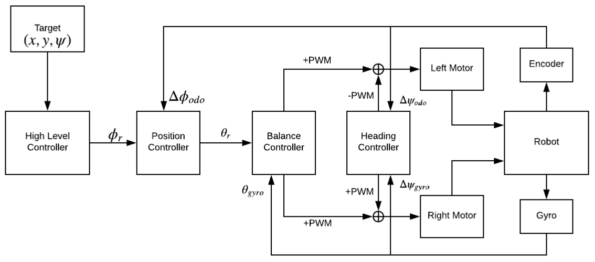


Fig. 2: Block Digram of the full control

$$\begin{aligned}
a_1 &= I_w + (m_r + m_w)R_w^2 \\
a_2 &= m_r R_w L \\
a_3 &= I_r + m_b L^2 \\
a_4 &= m_b g L \\
b_1 &= 2\tau_s \\
b_2 &= 2\tau_s/\omega_{NL} \\
c &= b_2(a_1 + a_2)s^2 \\
d &= (a_2^2 + a_1 a_3)s^4 \\
e &= b_2(a_1 + a_3 + 2a_2)s^3 a_1 a_4 s^2 a_4 b_2 s
\end{aligned}$$

$$D_1 = \frac{\Theta(s)}{U(s)} = \frac{c}{d+e} \tag{7}$$

$$D_2 = \frac{\Phi(S)}{\Theta(S)} = \frac{(a_2 + a_3)s^2 + a_4}{(a_1 + a_2)s^2} \tag{8}$$

C. Odometry

Our odometry calculates $(\Delta x, \Delta y, \Delta\psi, \Delta\phi)$ in between IMU interrupts, and we then integrate them to get values relative to the robot's starting position. We show parameters relavant to our odometry below:

Parameter	Meaning	Value
Δl	left encoder ticks	*
Δr	right encoder ticks	*
$\Delta\psi_g$	change in heading from gyro	*
d_l	left wheel diameter	0.08m
d_r	right wheel diameter	0.08m
b	effective distance between wheels	0.20m
g	gear ratio	20.4
r	encoder ticks per revolution	48
t_r	ticks to rotation conversion	1/gr
ψ_{thres}	gyrodometry difference threshold	0.005rad

TABLE III: Parameters for computing one step of our odometry process. We show input parameters to each step with the value being *.

Our odometry follows the process described in Algorithm 1 on every IMU interrupt (100Hz). It implements fusion of odometry and gyroscope measurements for calculating heading as described in [3].

Algorithm 1 Gyro-fused gyrodometry process used on every IMU interrupt.

```
$\Delta S_l = \Delta l \pi d_l t_r$
$\Delta S_r = \Delta l \pi d_r t_r$
$\Delta S = (\Delta S_l + \Delta S_r)/2$
$\Delta\psi = \frac{\Delta S_r - \Delta S_l}{b}$
$\Delta x = \Delta S \cos(\psi + \Delta\psi/2)$
$\Delta y = \Delta S \sin(\psi + \Delta\psi/2)$
$\begin{bmatrix} x \\ y \\ \phi \end{bmatrix} \leftarrow \begin{bmatrix} x \\ y \\ \phi \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\phi \end{bmatrix}$
if $|\Delta\psi_g - \Delta\psi| \geq \psi_{thres}$ then
    $\psi \leftarrow \psi + \Delta\psi_g$
else
    $\psi \leftarrow \psi + \Delta\psi$
```

We found the effective distance between wheels by manually turning (motor off) the robot in place and examining the odometry output. If the odometry calculates a turn greater than actual, then increase the estimated distance by the same multiplicative factor. Vice versa for an underturn.

Similarly, we determined the wheel diameters by driving the robot manually in a straight line around a meter stick and calcuating the diameter from the accumulated encoder ticks.

With more time we would have determined these parameters and found corrections against other errors using the UMBmark procedure [4].

We determined the ψ_{thres} gyrodometry parameter by looking at the difference in heading change measured by odometry and gyroscope in situations with very different wheel slippage seen in Figure 3. Gyrodometry relies on odometry when there is little slippage and on the gyro otherwise. We chose ψ_{thres} such that driving straight would almost always have odometry active while driving over wires (peaks of Figure 3 bot) would always have the gyroscope active.

D. High Level Controller

Our high level controller takes in a sequence of targets, tuples of the form $(x_d, y_d, \psi_d, type, duration)$ (desired state and type of target, from {TRANSLATE, ROTATE}). It runs in the setpoint thread at 25Hz and is only active when RC commands are absent. Its output is purely the setpoint that the lower level controllers stabilize to. The high level controller has the following parameters:

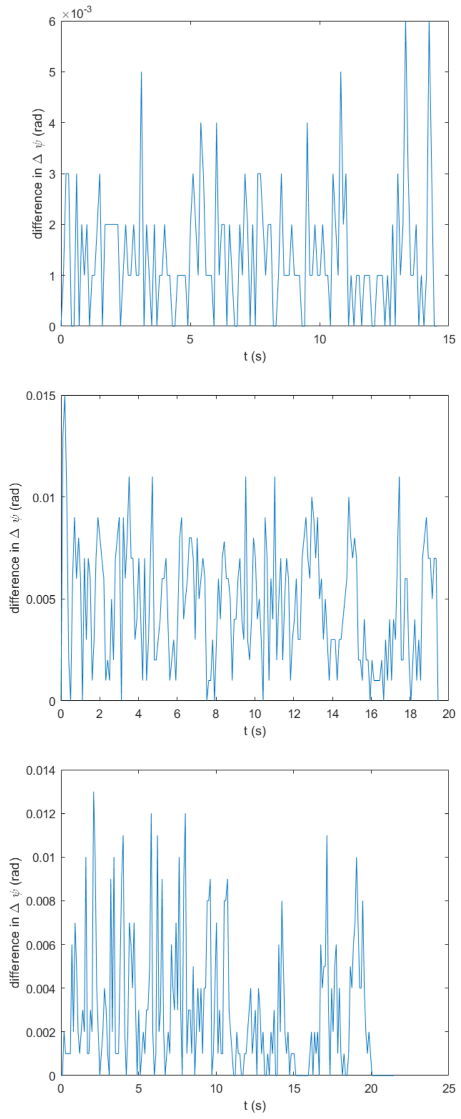


Fig. 3: Difference (absolute value) between change in heading measured by odometry and gyroscope when (top) driving straight with no slip, (mid) turning in place, and (bot) driving over wires.

parameter	meaning	value
setpoint	(ϕ, ψ) reference for inner controllers	**
targets	array of target structs	*
N	number of targets	*
t_ϕ	threshold for when ϕ is close enough	0.6rad
t_ψ	threshold for when ψ is close enough	0.05rad
d	average wheel diameter	0.08
t_1	threshold for too close to turn to target	5rad

TABLE IV: Parameters for the operation of the high level controller. * indicates an input parameter and ** indicates an output parameter.

We started with the angle thresholds as 0.001rad and gradually increased them until the robot converged to targets about 90% the time.

The high level controller follows Algorithm 2.

Algorithm 2 High level controller functions. The main function `Loop` is called in the setpoint thread at 25Hz. Parameters without a subscript such as ϕ are part of the state and globally accessible.

```
 1: global $i, t_i, \phi_i, \psi_i, \Delta\phi_i$
 2: $i \leftarrow 0$
 3: function ToTarget($t$)
 4:     $\Delta x = $ t.$x - x$
 5:     $\Delta y = $ t.$y - y$
 6:     $\Delta\phi = \sqrt{\Delta x^2 + \Delta y^2} * \frac{2}{d}$
 7:     $\Delta\psi = $ t.$\psi - \psi$
 8:     return $(\Delta\phi, \Delta\psi)$
 9: function StartTarget($t$)
10:     $t_i \leftarrow now()$
11:     $(\Delta\phi, \Delta\psi) = $ ToTarget(t)
12:     $\phi_i \leftarrow \phi$
13:     $\psi_i \leftarrow$ setpoint.$\psi$
14:     $\Delta\phi_i \leftarrow \Delta\phi + ($setpoint.$\phi - \phi)$
15: function Loop
16:     if $i < N$ then
17:         t = targets[$i$]
18:         $(\Delta\phi, \Delta\psi) = $ ToTarget(t)
19:         reached $\leftarrow$ (t.type == TRANSLATE and $|\Delta\phi| < t_\phi$) or (t.type == ROTATE and $|\Delta\psi| < t_\psi$)
20:         if reached == true then
21:             $i \leftarrow i + 1$
22:             StartTarget(targets[$i$])
23:         else
24:             p = $\max((now() - t_i)/$t.duration$, 1)$
25:             if t.type == TRANSLATE then
26:                 setpoint.$\phi \leftarrow \phi_i + p * \Delta\phi_i$
27:                 if $\Delta\phi > t_1$ then
28:                     setpoint.$\psi \leftarrow atan2(\Delta y, \Delta x)$
29:             else
30:                 setpoint.$\psi \leftarrow p *$ t.$\psi + (1 - p) * \psi_i$
```

At the start of each target we find out how much we have to travel in total to reach a new setpoint, then we blend the current setpoint to the new one over the duration of the target. At each time step, the setpoint the lower level controller sees will only differ by a small amount, even when moving onto the next target.

E. Navigation

Optitrack Motion Capture System

Odometry data drifts over time due to accumulation of errors (particularly from heading). Feedback from the Optitrack motion capture system provides ground truth to compare against and use in place of odometry.

This system, consisting of multiple cameras, is set up and calibrated with respect to the ground plane. The pose (x, y, ψ) of the Balancebot is tracked with an asymmetric rigid distribution of retroreflective tape markers on the Balancebot. The asymmetry resolves ambiguity in heading, and rigidity is required for recognition. A minimum of three markers is needed to ensure that the robot is being captured by at least one camera. The pose is transmitted to the Balancebot wirelessly through Zigbees. Communication between the receiver Zigbee and the Beaglebone is established through UART.

To determine the odometry's accuracy, we conduct an experiment to drive the Balancebot around a 1m square several times in both clockwise (cw) and counter-clockwise (ccw) direction. We compare the trajectories seen by the odometry against the motion capture ground truth, and is plotted in Figure 7 and 8.

Path Planning

We developed two planners for the Balancebot. The first is a Rotate, Translate, Rotate (RTR) planner to go between way points in a straight line. This is required to perform task 1. The second planner is based on the A^* search algorithm to generate motion commands based on the data from the motion capture system. This is essential for performing task 4 because there are obstacles (whose positions are not definite) and few constraints like opening gates in particular directions for that task, as described in the Introduction.

A^* Search Algorithm

An imaginary grid is visualized over the arena area. The cell size is chosen such that it is at least equal to the size of the Balancebot. Initially, two cells are defined, namely start and end cells. The start cell is the one in which the Balancebot is initially placed. From the motion capture system, the Balancebot's pose with respect to the ground plane are known. The start cell is determined from these. The end cell in this case is also the start cell because the Balancebot has to return back to the start pose after opening all the gates in the task.

The cells which contain the gates are marked as untraversable. Since the task requires the Balancebot to traverse through the gates for the first time in a particular direction, we differentiate the entry and exit positions of each gate. Given the start and end cells, the algorithm returns the optimal path between them by recursively exploring the map, assigning a cost to each cell and by updating them periodically. The total cost assigned to a cell is the sum of two costs, namely G cost (euclidean distance from start cell) and H cost (euclidean distance from end cell). The pseudo code is given in Algorithm 3.

Algorithm 3 Pseudo code for A^* Search Algorithm. The openList consists of cells yet to be evaluated, closedList contains cells which are already evaluated and the way points include entry points of all gates.

```
1: openList, closedList, startCell, endCell, wayPoints
2: ADD startCell to openList
3: function ASTAR
4:     function LOOP
5:         currentCell = cell in openList with lowest total cost
6:         REMOVE currentCell from openList
7:         ADD currentCell to closedList
8:         if currentCell == endCell then
9:             RETURN path
10:        for neighbours of currentCell do
11:            if neighbour == obstacle or neighbour in closedList then
12:                SKIP to next neighbour
13:        if neighbour is not in openList or new path to neighbour is shorter then
14:            SET total cost of neighbour
15:            SET parent of neighbour = currentCell
16:        if neighbor not in openList then
17:            ADD neighbour to openList
18: function MAIN
19:     Find startCell and endCell from Opitrack data
20:     Find waypoints of gates from Opitrack data
21:     Call function Astar to find optimal path through these points
```

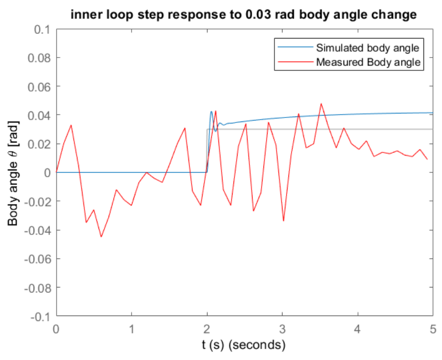
III. RESULTS

A. Motor Control

In order to model the system, knowledge about the parameters of motors like no load speed (ω_{nl}), motor constant (K), stall torque (τ_s), viscous friction coefficient (b), motor coil resistance (R), inertia (J) are necessary. These were determined by the methods explained in detail under Section II in Subsection II-A under 'Determination of Motor Parameters'. The measured as well as computed motor parameters are given in Table V.

Parameter with its unit of measurement	Left motor	Right motor
R (Ω)	6.1 ± 0.1	5.7 ± 0.1
$\omega_{nl}(rads/s)$	38.8367 ± 0.0247	39.6378 ± 0.0911
K (V s/rads)	0.2898 ± 0.1145	0.2809 ± 0.0795
τ_s (N m)	0.5544 ± 0.2197	0.5751 ± 0.1634
b	0.0005 ± 0.0074	0.0007 ± 0.0048
J (kg m_2)	0.0002 ± 0.0031	0.0003 ± 0.0055

TABLE V: Left and right motor parameters are displayed along with the confidence level of their measurements. The ± indicates the uncertainty in the measurements taken.



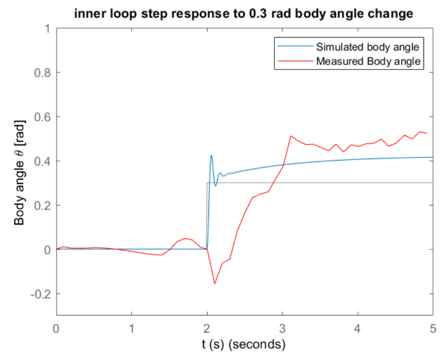
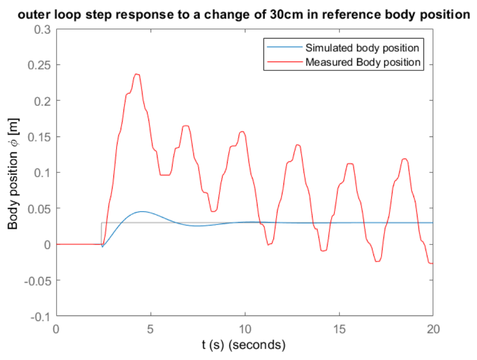


Fig. 4: Balance controller's response for (top) 0.03rad and (bot) 0.3rad step input in time domain.



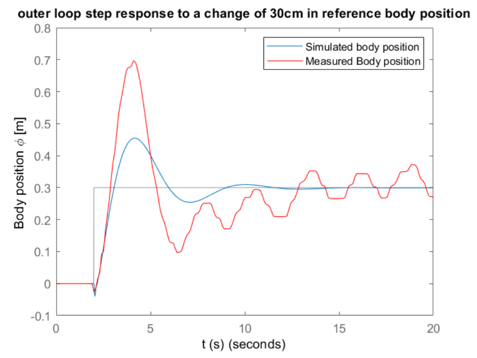


Fig. 5: Position controller's response for 3cm and 30cm step input in time domain

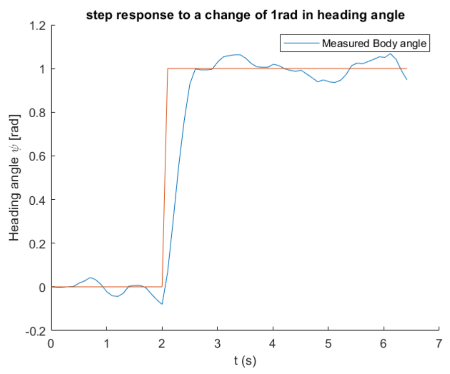


Fig. 6: Heading controller's step response for a 1rad input in time domain.

B. *Balance Control*

We subjected the Balancebot to step inputs through the balance and position controller to validate our model by comparing the actual and modelled responses. All responses are with the parameters in Table I. For the θ step response, the position controller was disabled to allow manual setting of θ_{set}. See Figure 4 for the balance controller step response to an input of 0.03rad and 0.3rad. With an initial $\theta_{set} = 0$, the balance controller kept the Balancebot standing still, oscillating with amplitudes of $\pm 0.04 rad$. After a 0.03rad perturbation, the body angle reached a new reference with a steady state error after 2s, but ϕ quickly diverged until the robot fell over. The measured body angle has 33% steady state error.

For the 0.3rad step input, the step response has a rise time 1s slower than the modelled response. The (apparent) steady state θ is 30% higher than simulated value. The robot fell over shortly after 5s so it is not actually steady state in reality.

The position controller was implemented and the step response for 3cm and 30cm (these were converted to anglees by dividing by average wheel diameter) are shown in Figure 5. In both cases, the actual overshoots were larger than the simulated response and gradually converged to the reference position. The overshoot could be caused by the large K_p value or small K_d Value. The heading controller converged to $\psi_{set} = 1$ with a rise time of 0.6s and no apparent steady state error. However, due to the oscillations of our balance controller heading still oscillates.

C. *Odometry*

We drove the Balancebot around a 1m side length square clockwise and counterclockwise (each 1-2 laps) under motion capture to quantify its accuracy seen in Figure 7 and 8. We characterize the accuracy of the odometry as a drift over distance travelled for position and heading.

For clockwise (cw)

$$d_{x,y} = \frac{0.12m}{20.8680m} = 5.8mm/m$$

$$d_{\psi} = \frac{0.1rad}{20.8680m} = 0.0048rad/m$$

For counterclockwise (ccw)

$$d_{x,y} = \frac{0.25m}{6.6637m} = 37.5mm/m$$

$$d_{\psi} = \frac{0.3rad}{6.6637m} = 0.045rad/m$$

The total distances travelled were calculated by integrating the distance between consecutive motion capture positions. We take the max directional drift as our accuracy, so $d_{x,y} = 37.5mm/m, d_{\psi} = 0.045rad/m$.

D. Path Planning

A sample arena shown in Figure 9 is considered. In the figure, 'G' stands for gate, 'L' for left post, 'R' for right post, 'S' for start cell, 'E' for end cell, 1 and 2 signify the gate number. The Balancebot has to return back to 'S' as the task requires. The path returned by the algorithm is marked in the figure and in terms of cell positions in the grid, it is [(3, 0), (3, 1), (3, 2), (2, 2), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 5), (2, 4), (3, 3), (3, 2), (3, 1)].

IV. Discussion

A. Balance Controller

We initially implemented a PID velocity controller for each motor such that the balance and heading controllers outputted a velocity instead of a PWM. The motivation for this was that each motor had different characteristics and to respond to changing voltage levels provided by the battery. This did not work because torque and velocity are non-linear so the same transfer functions would not hold with a scale factor, and because the additional controller slowed down the dynamics too much for the balance controller.

The performance of position and balance controller still need to be improved in terms of the oscillating behavior when standing and overshooting in position control. The oscillation could be eliminated by increasing the damping term in K, and the overshoot could be reduced by tuning the proportional and damping terms.

B. Odometry

We didn't have enough time to perform the full UMBmark procedure to quantify the accuracy of our odometry, which would have allowed us to apply corrections for systematic errors.

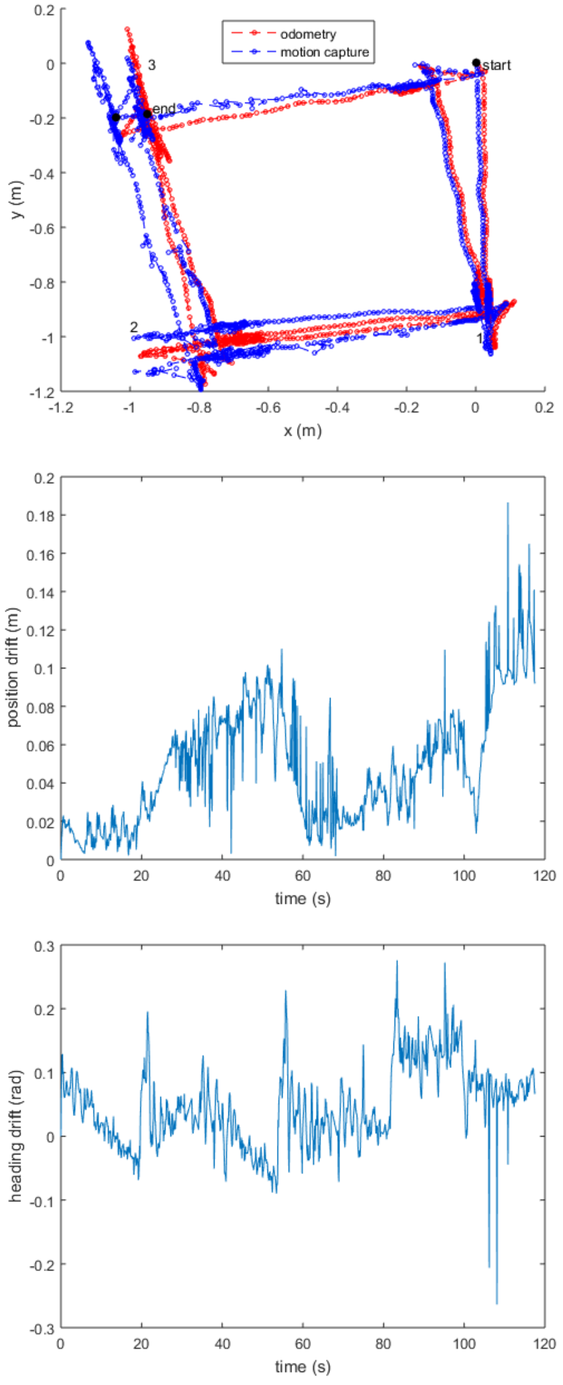


Fig. 7: (top) Clockwise trajectory of Balancebot around a 1m square with odometry compared against motion capture. The Balancebot travelled 1 lap and 3/4 of another lap. The motion capture coordinates were offset such that the robot started at the origin, and the odometry points were rotated by -0.046rad so that the inital headings are aligned. We also show the (mid) position drift and (bot) heading drift over time.

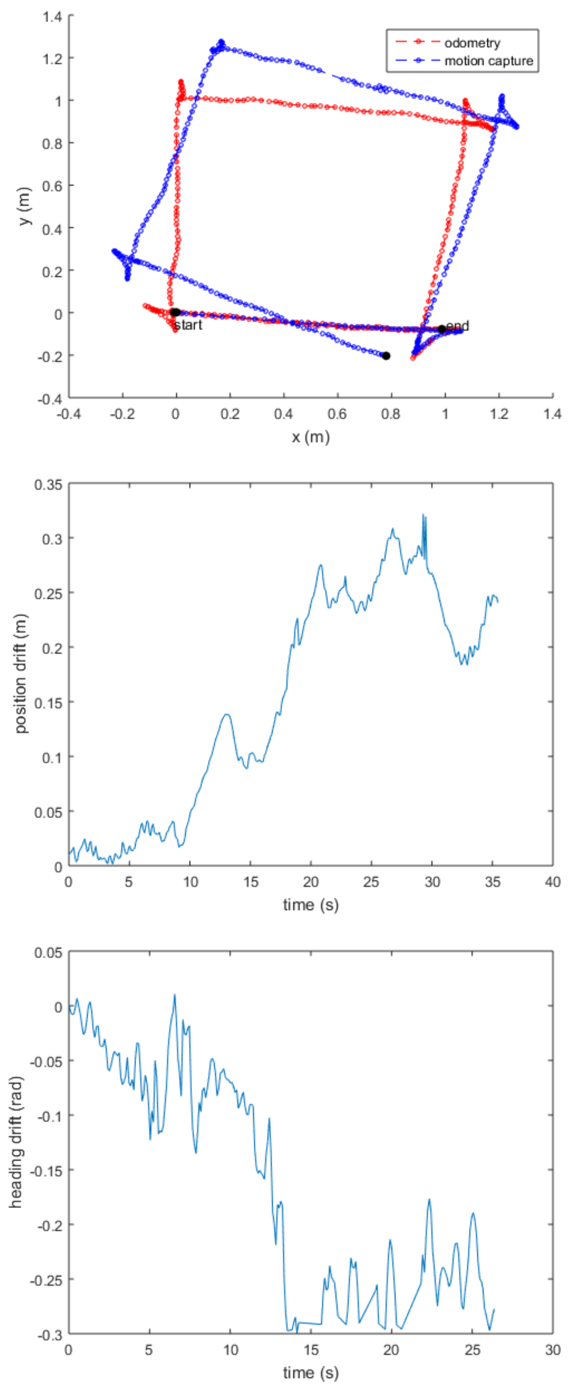


Fig. 8: (top) Counterclockwise trajectory of Balancebot around a 1m square with odometry compared against motion capture. The Balancebot travelled 1 lap and 1/4 of another lap. The motion capture coordinates were offset such that the robot started at the origin, and the odometry points were rotated by 0.076rad so that the inital headings are aligned. We also show the (mid) position drift and (bot) heading drift over time.

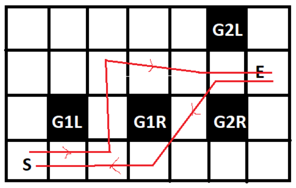


Fig. 9: Sample arena fed to A^* search algorithm.

The total distance travelled in the cw direction was a lot more than the ccw direction because it took much longer to converge around its targets, driving back and forth in a straight line around them. This made up a significant part of the accumulated 20.8680m. Since drift is mostly accumulated during slipping [4], which is negligible during straight driving in our case, this could explain why the drift was apparently much less than for the ccw direction.

Another possible factor is the combination of different sources of error, such as unequal wheel diameter and unequal loading across the wheel axis, adding up in the ccw direction while cancelling out in the cw direction.

References

[1] P. Gaskell, “Lecture on balancebot controllers,” October 2018.

[2] O. S. M. N. B. T. Gaskell, P., “Balancebot dynamics handout,” September 2018.

[3] J. Borenstein and L. Feng, “Measurement and Correction of Systematic Odometry Errors in Mobile Robots,” Tech. Rep. 6, 1996. [Online]. Available: http://www-personal.umich.edu/ johannb/Papers/paper58.pdf

[4] ——, “Measurement and Correction of Systematic Odometry Errors in Mobile Robots,” Tech. Rep. 6, 1996. [Online]. Available: http://www-personal.umich.edu/ johannb/Papers/paper58.pdf