AN INTRODUCTION TO THE HOLONOMIC BALLBOT AS A DYNAMICAL SYSTEM

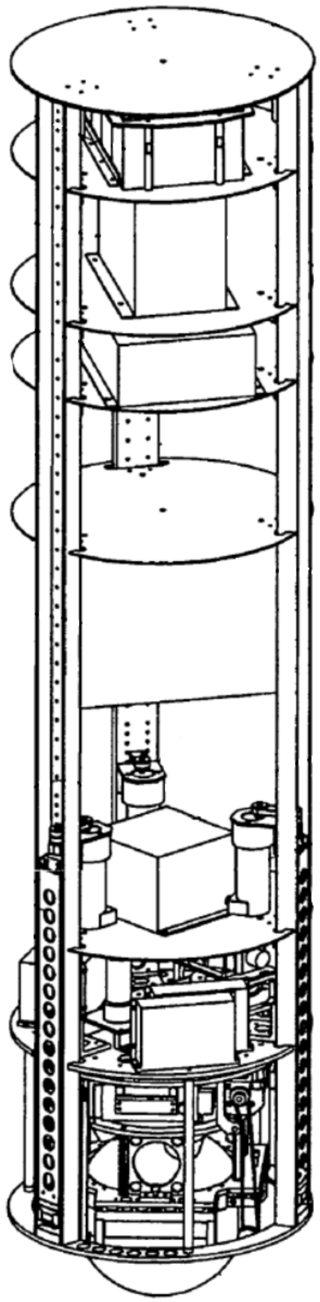

Figure 1: Patent 7,847,504 B2 [1], modified to remove labels

Miami University
Oxford, Ohio
2025

Samuel Skinner, Author
Dr. Van Scoy, Professor

Table of Contents

1

Introduction to the Holonomic Ballbot as a Dynamical System

The ballbot is a recently patented [1] robot with a unique structure consisting of a spherical base combined with a tall, narrow, body as shown in Figure 1. Unlike other, statically stable robots , the ballbot is inherently unstable. However,its unique shape make it of significant interest for modern robotics and agile navigation within flat environments, despite its currently limited uses.

Physically, the ballbot consists of two moving and interlinked parts: the ball and the body. The ball is a sphere which remains in constant contact with the ground while the body is an elongated cylinder which houses the control hardware , power supply, and balances on top the ball. In practical application, there are typically three points of contact between the base and the ball, each of which is attached to a motor and allow the body to drive the wheel and, in turn, allow the robot to balance and move.

From a controls perspective, the ballbot is related to a 3D inverted pendulum. More specifically, the ball is modeled as a uniformly dense sphere which is attached to a uniformly dense cylinder. This system is inherently unstable under open-loop conditions, having one unstable equilibrium when the ballbot is perfectly vertically aligned. Without active control, in a non-idealized environment, the ballbot will rapidly diverge from this equilibrium and fall.

As will be described in subsequent chapters, we define the ballbot as having two inputs representing perpendicular torques on the ball alongside 8 states, representing positions, angles, velocities and angular velocities of both the ball, and relation to it. Constructing these two torques from the practical three motor input is possible, but will not be discussed here. We will also assume an environment without slipping between the ground and the ball. Finally, we will not be modeling the yaw rotation of the body nor the rotation of the ball itself, as both create a non-holonomic system rendering it unable to be stabilized using static state feedback [2] as discussed in class.

2

Modeling

Although a ballbot is a fairly complex system, it can be thought of in various ways. The most useful of which, will be as a inverted pendulum affixed atop a freely translatable card. It's states can be seen from inspection of its movement, while state equations will be derived using lagrangian mechanics to derive a continuous nonlinear time-invariant system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, u)$.

Upon inspection we can note the following states, additionally shown in 2.1.

- x positional states: x and $\dot{x}$
- y positional states: y and $\dot{y}$
- Pitch states: θ and $\dot{\theta}$
- Roll states: ϕ and $\dot{\phi}$

Additionally, we define the following constants that parametrize the simulation, after each a typical value is given

- Radius of the ball: r (0.1 m)
- Distance between the ball and the center of mass of the body: d (0.5 m)
- Mass of the ball: m_{ball} (2 kg)
- Mass of the body: m_{body} (10 kg)
- Moment of the ball around all axes: I_{ball} (0.008 kgm^2)
- Moment of the body around the x and y axes: I_{bodyxy} (0.8725 kgm^2)
- Moment of the body around the z axis: I_{bodyz} (0.078125 kgm^2)
- Force of gravity: g (10 m/s^2)

Finally, we define the following inputs to the system.

- Ball x Torque: τ_x
- Ball y Torque: τ_y

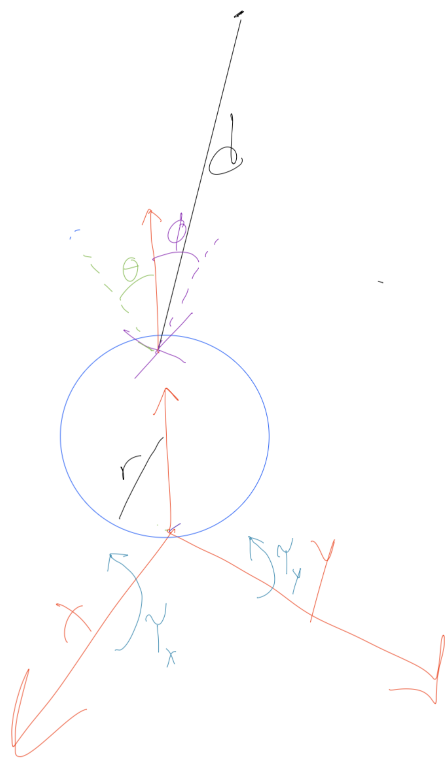

Figure 2.1: Drawing of a Ballbot with states x, y, θ, and ϕ visible

2.1 Definition of States

These states, are grouped into a set of generalized coordinates, $q = [x, y, \theta, \phi]^\top$ and then combined to form the full state of the system

$$\boldsymbol{x} = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \\ \phi \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix}.$$

The inputs are represented as $u = \begin{bmatrix} \tau_x \\ \tau_y \end{bmatrix}$.

2.2 Derivation of State Derivatives

A general nonlinear time-invariant system is defined as $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, u)$, however for the purposes of this document, it is more helpful to think in the form

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = g(q, \dot{q}, u)$$

.

Upon inspection, one can see that both sides of the equation share the state $\dot{q}$, showing that one just needs to find a function $\ddot{q} = f(q, \dot{q}, u)$ which models the dynamics of the system as the $\dot{q}$ states can simply be passed through in f.

The derivation of this function g will be achieved through the use of lagrangian mechanics. Lagrangian mechanics constructs a function, called the lagrangian defined as the difference between the kinetic energies T and potential energies U of a system:

$$\mathcal{L} = T - U$$

For the ballbot system, the total kinetic energy is the sum of the translational and rotational energy of both the ball and the body. Meanwhile, the total potential energy is defined as the sum of the gravitational potential energy of the ball and the body.

By combining this, we can state the full lagrangian as: $\mathcal{L} = (T_{ball,translation} + T_{ball,angular} + T_{body,translation} + T_{body,angular}) - (U_{ball,height} + U_{body,height})$

2.2.1 Derivation of translational energies

The translational kinetic energy of any rigid body is defined by the velocity of its center of mass. While the scalar form $E = \frac{1}{2}mv^2$ is familiar, in the context of this document, it will extended into the vector case, written more generally as

$$T = \frac{1}{2}mv^\top v$$

In our case, defining these velocities is tricky, especially for the body, so we will rely on the fact that the time derivative of the position vector (p)is the velocity vector. This final equation is written:

$$T = \frac{1}{2}m\dot{p}^\top \dot{p}$$

Translational energy of the ball. Starting off with the position of the ball, it can be easily seen that the position of the center of mass of the ball will be the sum of the position of it's base plus the radius of the ball upwards.

$$p_{ball} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} + \begin{bmatrix} x \\ y \\ r \end{bmatrix} = \begin{bmatrix} x \\ y \\ r \end{bmatrix}$$

Taking the time derivative of this vector yields

$$v_{ball} = \dot{p}_{ball} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ 0 \end{bmatrix}$$

Finally, this can be inputted into the translational energy formula to arrive at

$$T_{ball,translation} = \frac{1}{2} m_{ball} v_{ball}^{\top} v_{ball}$$

Translational energy of the body. The position of the body's center of mass is more complex as it depends not only on the position of the ball, but also on the angles of the body (θ and ϕ) and the distance between the ball's center and the body's center of mass (d)

However, the rotation of the body is defined by the rotation matrix sequence of rotating around the X axis an angle ϕ followed by a rotation around the Y axis an angle θ. These orations can be codified in the rotation matrices which map outwards from a local to a global frame

$$R_{body} = R_y(\theta) R_x(\phi) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}.$$

By using these matrices, one can easily calculate the position of the center of mass of the body as

$$p_{body} = p_{ball} + R_{body} \begin{bmatrix} 0 \\ 0 \\ d \end{bmatrix}$$

Therefore, using the same method as established previously, one can calculate the velocity as the time derivative of the position vector. Do note that, because of the chain rule, and the resulting expression being dependent on θ and ϕ that the jacobian should be used in the form

$$v_{body} = \frac{dp_{body}}{dt} = \frac{\partial p_{body}}{\partial q}\frac{\partial q}{\partial t}$$

Resulting in a similar relation of

$$T_{body,translation} = \frac{1}{2} m_{body} v_{body}^{\top} v_{body}$$

2.2.2 Derivation of angular energies

In a similar method as in 2.2.1, one can start with the equation for the angular energy and similarly extend it to the vector case. The scalar form is $E = \frac{1}{2}I\omega^2$ where I is the moment of inertia of the spinning body and ω is the angular velocity of that body. In the vector case, this similarly transforms into

$$E = \frac{1}{2}\omega^{\top} * I * \omega$$

. Where I is the inertia matrix of the system. For our application,

Rotational energy of the ball. Since the ball never slips on the ground, it's angular velocity is directly derived from the linear velocity of the ball. For a ball of radius r, motion in the +x axis, induces a rotation around the +y axis. And, similarly, motion in the +y axis induces a rotation round the -x axis because of the right handed axes being used. Therefore, because of $\tau = rF$ restated as $F = \tau/r$, one can see that

$$\omega_{ball} = \begin{bmatrix} -\dot{y}/r \\ \dot{x}/r \\ 0 \end{bmatrix} \tag{2.1}$$

which results in the final energy equation based on the scalar inertia of the ball since it is uniform regardless of axis

$$T_{ball,angular} = \frac{1}{2}\omega_{ball}^{\top} I_{ball} \omega_{ball}$$

.

Rotational energy of the body. The body's angular velocity, ω_{body} is similarly calculated from the time derivatives of the body angles. Do note that the angles must end up in the local frame of the body with respect to the global axes. The easiest way to do this is to bring all angles into the global coordinate frame, before transforming them back into the local space of the body. Doing so, yields the equation

$$\omega_{body} = R_{body}^{\top}(\begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_y(\theta)\begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix})$$

and the energy relation

$$T_{body,angular} = \frac{1}{2}\omega_{body}^{\top} * I_{body} * \omega_{body}$$

where $I_{body} = \begin{bmatrix} I_{bodyxy} & 0 & 0 \\ 0 & I_{bodyxy} & 0 \\ 0 & 0 & I_{bodyz} \end{bmatrix}$.

2.2.3 Derivation of potential energies

The potential energies of the ball and the body can be calculated from the formula for gravitational potential energy of a point mass of mass m at a height h with gravitational acceleration g as $E = m * g * h$.

Since we already calculated the position vectors of the ball and the body, we can just access their z component and use that in the formula. Mathematically, this can be stated by taking the dot product with the third cartesian basis vector $\hat{k}$.

Therefore the potential energy for both the ball and body can be expressed as

$$U_{ball,height} = m_{ball}g(p_{ball} \cdot \hat{k})$$

$$U_{body,height} = m_{body}g(p_{body} \cdot \hat{k})$$

2.3 Formation of the equations of motion

Using the relation derived above for the lagrangian, we now have the lagrangian and can use it to derive our equations of motion. The equations of motion are derived using the Euler-Lagrange equation, which for this document will be written as

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}} - \frac{\partial \mathcal{L}}{\partial q} = \mathcal{Q}$$

where $\mathcal{Q}$ is the generalized input forces of the system.

Generalized Forces. In our case, we have two input forces are τ_x and τ_y which require a transformation into our generalized coordinates q. As each affects both the rotation of the body angles and causes some push back from $F = \tau/r$, we need a matrix that maps between our inputs u and $\mathcal{Q}$. The matrix that maps between this is

$$\mathcal{Q} = B_{inputtogeneralized} * u = \begin{bmatrix} 0 & -1/r \\ -1/r & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \tau_x \\ \tau_y \end{bmatrix}$$

and is derived using the same idea as was used in equation 2.1

Equations of motion. Now that we have a representation of the lagrangian and the generalized input forces, this system can be solved symbolically to return the function $\ddot{q} = g(q, \dot{q})$ that we originally sought.

Finally, the state update equation we sought $\dot{\boldsymbol{x}}$ can be calculated

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{q} \\ g(q, \dot{q}) \end{bmatrix}$$

3

Ballbot Simulation

The following sections visually demonstrate the uncontrolled dynamics of the ballbot under various initial conditions.

3.1 No Deviation

The following are graphs show the ballbot stand perfectly still at its unstable equilibrium point, with initial state $\boldsymbol{x} = 0$. Both graphs are as expected, as although the equilibrium point is unstable, no deviations are made to cause the ballbot to move from it.

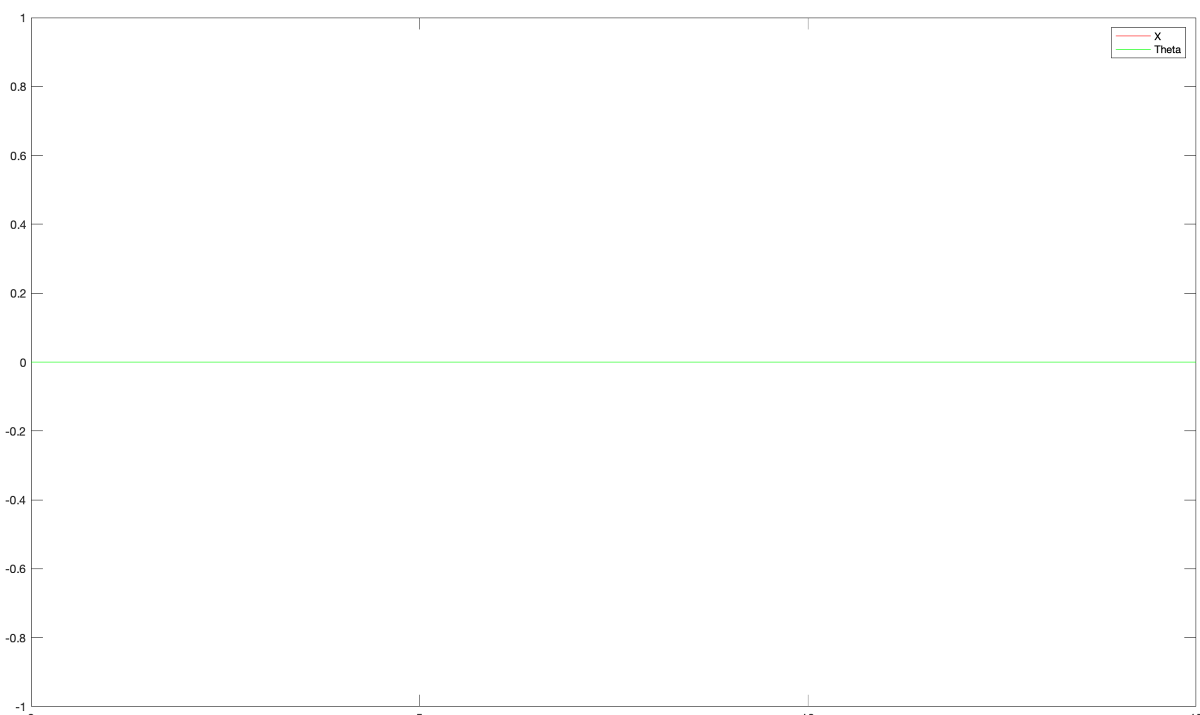

Figure 3.1: Graph of the x and θ coordinates of the ballbot at its equilibrium point

3.2 Theta Deviation

Figures 3.2 and 3.3 represent the ballbot starting with $\boldsymbol{x} = \begin{bmatrix} 0 & 0 & 0.00001 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top$. These behave mostly as expected, with two things of note. First, is the slight positive x drift in 3.2, we assume this to be the result of simulation inaccuracy. Second, is the increasing angle of theta, which is odd but not a problem as we are not attempting to control the system. A more rigorous simulation would have something in place to prevent this.

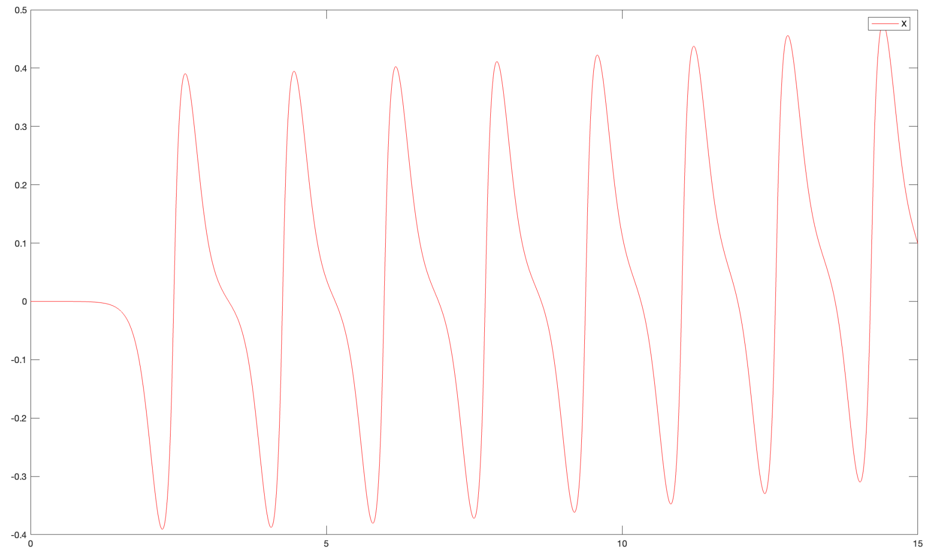

Figure 3.2: Graph of the x coordinate of the ballbot starting near its equilibrium state with a slight deviation to θ

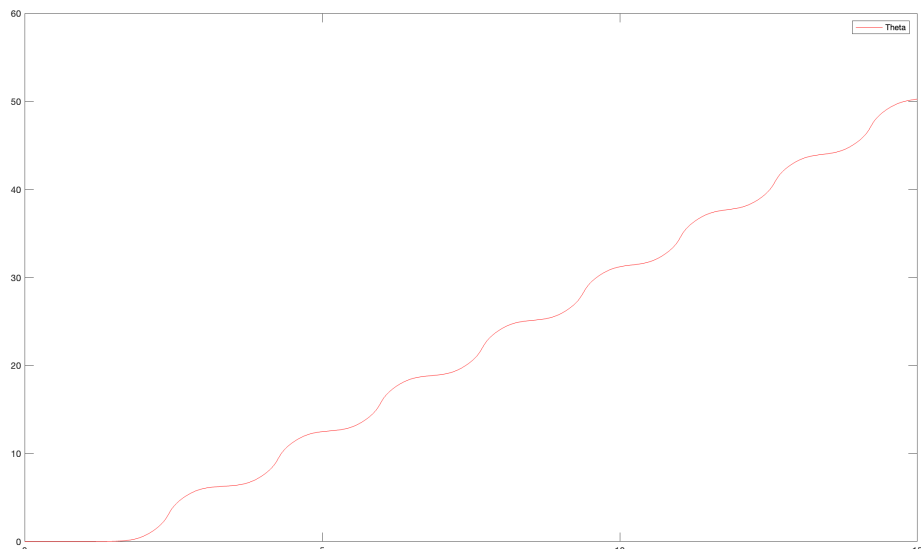

Figure 3.3: Graph of the θ coordinate of the ballbot starting near its equilibrium state with a slight deviation to θ

3.3 Theta and Phi Deviation

Figures 3.4, 3.5, 3.6, and 3.7 start with $\boldsymbol{x} = \begin{bmatrix} 0 & 0 & 0.00001 & 0.0001 & 0 & 0 & 0 & 0 \end{bmatrix}^\top$. At these initial conditions we can start to notice sone chaotic behavior arising from the interaction between the nonlinear pendulum force and the linear kickback of the wheel from the falling body.

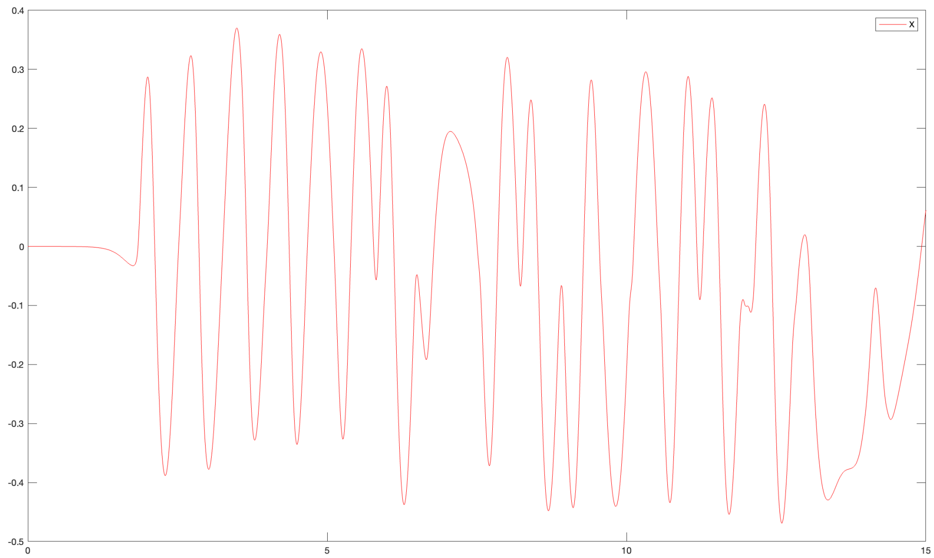

Figure 3.4: Graph of the x coordinate of the ballbot starting near its equilibrium state with a slight deviation to θ and ϕ

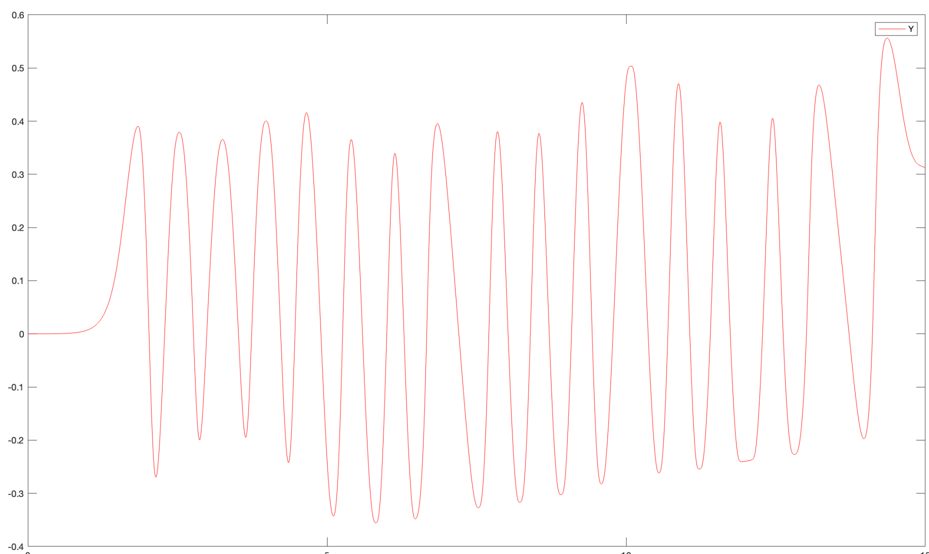

Figure 3.5: Graph of the y coordinate of the ballbot starting near its equilibrium state with a slight deviation to θ and ϕ

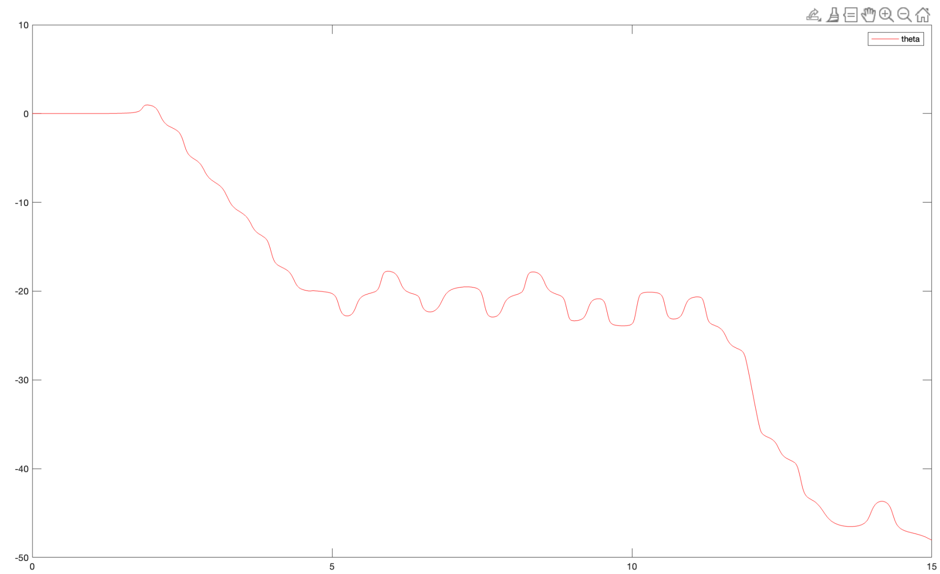

Figure 3.6: Graph of the θ coordinate of the ballbot starting near its equilibrium state with a slight deviation to θ and ϕ

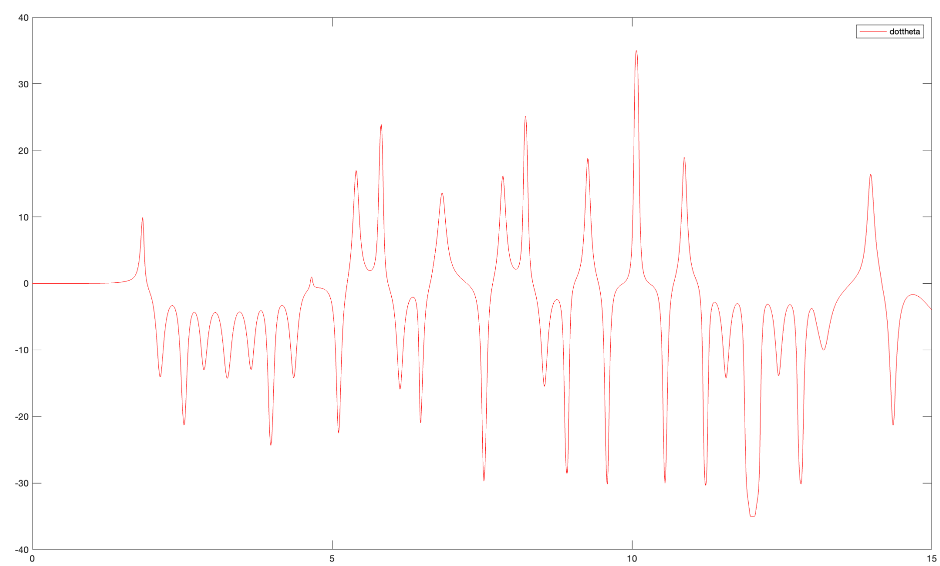

Figure 3.7: Graph of the $\dot{\theta}$ coordinate of the ballbot starting near its equilibrium state with a slight deviation to θ and ϕ

4

Linearization

Linearization is the method of taking a time-invariant continuous-time system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, u)$ and approximating it as a linear time-invariant system around some state and input of the form

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + Bu$$

A linearization is calculated by evaluating the jacobian of the function f with respect to the state or input. Put symbolically, given an equilibrium state $\tilde{\boldsymbol{x}}$ and an equilibrium input $\tilde{u}$,

$$A = \frac{\partial f}{\partial \boldsymbol{x}}(\tilde{\boldsymbol{x}}, \tilde{u})$$

and

$$B = \frac{\partial f}{\partial u}(\tilde{\boldsymbol{x}}, \tilde{u}).$$

For the ballbot, this means calculating its equilibrium points and the jacobian of the full state update function.

4.1 Equilibrium points of the ballbot

Equilibrium points are defined as states where the system dynamics are stationary, implying

$$\dot{\boldsymbol{x}} = f(\tilde{\boldsymbol{x}}, \tilde{u}) = 0$$

From the definition of f as calculated before, one can see that any equilibria point of the ballbot will be dependent on the state and the input. Looking first toward the state $\boldsymbol{x} = \begin{bmatrix} x & y & \theta & \phi & \dot{x} & \dot{y} & \dot{\theta} & \dot{\phi} \end{bmatrix}^\top$

In order for $\dot{\boldsymbol{x}} = 0$ to remain true, the state must have its angles (θ and ϕ), translational velocities ($\dot{x}$ and $\dot{y}$), and angular velocities ($\dot{\theta}$ and $\dot{\phi}$) all be equal to zero. Interestingly, this means that the ballbot can be linearized around any vertical position, however, as we will see shortly, this will not affect the output A and B matrices.

Moving onto the inputs, it's trivial to see that in order for $\dot{\boldsymbol{x}} = 0$ to hold, $u = 0$ must be true. As any input would immediately knock the system out of the equilibria point.

4.2 Resultant A and B matrices

Using the above equations, one can pick a value for $\tilde{\boldsymbol{x}}$ and use it to calculate the A and B matrices. Doing this symbolically in Matlab, when combined with the derivation steps as derived earlier, results in the following matrices for the choices of $\tilde{\boldsymbol{x}} = 0$ and $\tilde{u} = 0$. Note the inclusion of the shared denominator defined $D = I_{ball}I_{xy} + I_{ball}m_{body}d^2 + I_{xy}m_{ball}r^2 + I_{xy}m_{body}r^2 + m_{ball}m_{body}r^2d^2$.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{m_{body}^2 g r^2 d^2}{D} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{m_{body}^2 g r^2 d^2}{D} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{m_{body} g d (I_{ball} + m_{ball} r^2 + m_{body} r^2)}{D} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{m_{body} g d (I_{ball} + m_{ball} r^2 + m_{body} r^2)}{D} & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -\frac{r(I_{xy} + m_{body}d^2 + m_{body}rd)}{D} \\ \frac{r(I_{xy} + m_{body}d^2 + m_{body}rd)}{D} & 0 \\ 0 & \frac{I_{ball} + m_{ball}r^2 + m_{body}r^2 + m_{body}rd}{D} \\ \frac{I_{ball} + m_{ball}r^2 + m_{body}r^2 + m_{body}rd}{D} & 0 \end{bmatrix}$$

As expected from the equilibria derivation, the first two columns of the linearization are zeros implying that the linearization is independent of the state's x and y. Also, we see the same one to one mapping of the $\dot{q}$ vectors in the upper right corner of the matrix as in f

5

Estimation

For feedback control to be implemented, one needs to have a measurement of the entire state, $\boldsymbol{x}$, of a system. When working with a purely mathematical simulation, this is trivial as one can simply access the desired state and use it. However, in practical application, the full state is often not directly available. Sensors can only sense some states, therefore some type of estimation is needed to model the impossible to measure states in terms of the possible to measure ones.

The observability of a system is wholly defined as a function of the C matrix and this can be easily seen from imagining an empty C matrix and an identity C matrix with the same size as A. With this in mind, the number of observable states can be determined by the rank of the observability matrix

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}.$$

5.1 Observability of the ballbot

In the context of a ballbot without external tracking, there is no sensor that can measure x, y without an external reference. $\dot{x}$ or $\dot{y}$ can be approximated via the use of optical sensors such as those in the bottom of a computer mouse. Finally, θ, ϕ, $\dot{\theta}$ and $\dot{\phi}$ are the most measurable states because they can be measured with a gyroscope. One should note here that $\ddot{x}$ and $\ddot{y}$ can be measured by the use of an accelerometer, but those are not in the state and therefore not helpful to us as we are not in the realm of practical application.

With this in mind, we will look at some choices of outputs and see what they imply with regards to the observability of the system as a whole.

Case 1: Position states x and y. Defining $C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ we can solve for $rank(\mathcal{O})$ and find it to be 8, implying that the system is fully observable. Despite this theoretical result, we were unable to get a simulation working with this restricted set of outputs.

Case 2: Gyroscope measureable states θ, ϕ, $\dot{\theta}$ and $\dot{\phi}$. By following the same method as above, defining $C = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$ we can find the $rank(\mathcal{O})$ and determine it to be 6. Upon inspection of the resultant matrix, one will find that only the x and y position states to be not observable. While this seems like a useless result at first, this combination of outputs (sensors) and observable states makes it the perfect C matrix if we were creating a robot that needed to self balance.

Case 3: Hybrid states x, y, $\dot{\theta}$ and $\dot{\phi}$. For the purposes of this document, we have chosen the output matrix to be $C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$. This choice creates a balance where the system is both fully observable, is actually able to be simulated, and makes the observer fulfil a purpose as there are some states whose values can only be estimated through the use of the observer. Since we will need to know the precise x and y positions for our desired simulation, we cannot get around the fact that we need those states to be visible in the output. It should be stated that this choice of C has no specific physical interpretation, this is just a nice middle ground set of outputs that is neither trivial nor extremely difficult.

5.2 Design of the Observer

For this document, we will use a standard Luenberger Observer taking the form

$$\begin{aligned}\dot{\hat{\boldsymbol{x}}} &= f(\hat{\boldsymbol{x}}, u) - L(y - \hat{y}) \\ \hat{y} &= C\hat{\boldsymbol{x}}\end{aligned}$$

where L is the observer's gain. To select the observer's gain, we will use matlab's lqr function, passing in the linearization $A^\top$ and $C^\top$. Selecting the gains was done by trial and error and eventually resulted in state gains of $Q = 10 \cdot \mathrm{diag}(1, 1, 25, 25, 1, 1, 5, 5)$ and input gains of $R = I_{4x4}$ which we found to work well for our application.

After execution and transposing to get our proper form, the resultant gain matrix is

$$L = \begin{bmatrix} 4.3068 & 0 & -0.4276 & 0 \\ 0 & 4.3068 & 0 & 0.4276 \\ -0.0898 & 0 & 16.8427 & 0 \\ 0 & 0.0898 & 0 & 16.8427 \\ 4.3657 & 0 & -13.7123 & 0 \\ 0 & 4.3657 & 0 & 13.7123 \\ -0.4276 & 0 & 35.1644 & 0 \\ 0 & 0.4276 & 0 & 35.1644 \end{bmatrix}$$

6

Control

The goal for this project was to create a demonstration of the ballbot moving in a figure eight pattern. More specifically, this means, starting the initial $\boldsymbol{x}$ at some arbitrary position and angle and with an observer initialized to $\hat{\boldsymbol{x}} = 0$ Since the observer has already been defined, we still need to define the controller and the reference function for the figure eight's path.

The path traced by a figure eight is nontrivial and requires a balance between position and angle. If the controller focuses too much on the position, the robot will move too quickly and fall over. On the other hand, if the controller focuses too much on angle, it won't trace out the correct path.

6.1 Definition of the reference function

A reference signal is function that evaluates to a desired state of the system. First, recall that the state vector is $\boldsymbol{x} = \begin{bmatrix} x & y & \theta & \phi & \dot{x} & \dot{y} & \dot{\theta} & \dot{\phi} \end{bmatrix}^\top$. Second, we can see, upon inspection that the only states which matter in a figure eight pattern are the x and y states of the base of the robot.

In order to derive the equation for this path, one can first start with the parametric form of a circle $(x, y) = (r * cos(\omega t), r * sin(\omega t))$ and what is a figure eight if not a circle where one component is twice the frequency of the other? This yeilds the final equation $(x, y) = (r * cos(\omega t), 0, 5 * r * sin(2 * \omega t))$

Finally, this can be written in the form of a function in matlab to represent

$$\begin{bmatrix} r * cos(\omega t) \\ 0.5 * r * sin(2 * \omega t) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

6.2 Controller

In order to achieve this goal, we will use LQR and use gains that put a stronger weight on the x and y components in comparison to the other states, but not have them be too large as to cause the robot to fall over.

By trial and error, the gains we selected were $Q = 10 \cdot \text{diag}(10, 10, 1, 1, 1, 1, 1, 1)$ and $R = I_{2x2}$ as they appeared to strike a nice balance between acceptable tracking of the reference vector without the robot falling over.

By using the built-in matlab function, I got a K matrix of

$$K = \begin{bmatrix} 0 & -3.1623 & 0 & 32.1665 & 0 & -4.2961 & 0 & 6.9921 \\ 3.1623 & 0 & 32.1665 & 0 & 4.2961 & 0 & 6.9921 & 0 \end{bmatrix}$$

This is used in stoic state feedback with the system with the input vector u being calculated as

$$u = -K(\hat{\boldsymbol{x}} - r)$$

6.3 Results

results with a couple of graphs, do x and y position vs time

References

[1] Ralph L. Hollis. Dynamic balancing mobile robot, U.S. Patent 7847504 B2, December 7 2010.

[2] Yasir Awais Butt. Robust stabilization of a class of nonholonomic systems using logical switching and integral sliding mode control. *Alexandria Engineering Journal*, 57(3):1591–1596, 2018.