ROB 101: Project 3, Segway

Eva Mungai, Grant Gibson, Wami Dosunmu-Ogunbi, and Jessy Grizzle

Fall 2020

1 Project Introduction

Picture this: You are a new member of an up and coming competitive tag group called The Cassie Bots. It is your turn in the arena, and you are nervous because you do not want to let your teammates down. You observe your opponent intently, already formulating strategies of where and when to move to not get tagged. The buzzer rings, indicating that the round has begun. Your opponent lunges at you, and you evade. As she jumps over barriers and takes jabs at you, you duck, roll, and shimmy (yes shimmy because your moves are on point) out of the way. Your mind is racing, adjusting and readjusting strategies in accordance to how your opponent reacts. You start down one path, only to change course last minute when you see your worthy opponent make a move that would have ultimately cut you off.

Figure 1: Competitive tag

Besides being such a cool way to legitimize grown adults playing a childish game, competitive tag provides quite the helpful illustration to the next major topic that we would like to cover in this class: **Model Predictive Control** (MPC). (If you have never seen competitive tag before it's pretty intense! Check out a clip here if you are interested.)

The premise of MPC in the context of the competitive tag example is this: You have information on your past (the previous moves that you made to evade your opponent as well as the moves that she has made to try to best you) and you know your current state (yours and your opponent's current position). Using this knowledge, you plan a path for you to take to evade your opponent's next move.

In other words, in MPC you use information on your past and present to predict what would happen in the future so that you can act appropriately now. Because this is your first introduction to the subject, we will of course scale things back a bit and not deploy the full power of Model Predictive Control. However, we'll still give you a taste of the subject and then perhaps you'll want to learn more!

In this document, we will show you how to model a self-driving car moving along a straight path. Using what you have learned earlier in this semester, we will show you how to transform an MPC problem to look like a special case of **least squares for underdetermined systems of linear equations**. When we told you that **least squares was a super power**, you may have blown it off as hyperbole. Well, when you are done with this project, you may change your tune!

The car is just a warm-up act before the main band comes on the set, the Segway. We will introduce to you a simplified model of a Segway where you will take what you learn from the self-driving car example to design your own Segway balancing controller.

2 Car on a Straight Path

Imagine that you've landed your dream internship at a self-driving car company (lucky you!). Your first task on the job is to design an automatic throttle control system [1] for the car. As the car is a prototype, it is programmed to only drive in a straight line. For any other intern this might be a daunting task but not for you. Armed with your ROB 101 knowledge you will be ready! Let us take a look at how we can solve this problem. The first question at hand is how to describe the motion (or dynamics) of the self-driving car.

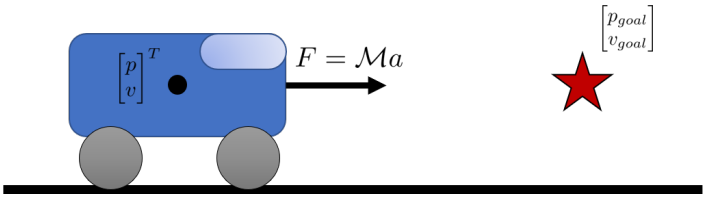


Figure 2: Self-driving car on flat ground.

2.1 Equations of Motion and Differential Equations

Ordinary Differential Equations for Year-1 Students

A ROB-101 appropriate introduction to the topic of ODEs is given in Appendix C of our textbook.

Newton's second law tells us that the force (F) applied to an object with mass ($\mathcal{M}$) is equivalent to its mass times acceleration (a). In other words, $F = \mathcal{M}a$. Physics and Calculus both tell us that acceleration is the rate of change of velocity (v) with respect to time (t), in other words,

$$\frac{dv(t)}{dt} = a(t).$$

If we plot velocity versus time, the acceleration at time t corresponds to the local slope of the plot at time t,

$$\frac{v(t+h) - v(t)}{h} \approx \frac{dv(t)}{dt} = a(t) \tag{1}$$

Next, to make these equations appropriate for a computer, we discretize time by defining

$$\begin{aligned} t_k &:= kh \\ v_k &:= v(kh) \\ a_k &:= a(kh). \end{aligned}$$

With these definitions in hand, we can be rewrite (1) as

$$v_{k+1} = v_k + ha_k. \tag{2}$$

We have converted the differential equation (1) into a difference equation, namely (2), that we can iteratively solve on a computer. We now have an approximate **finite difference equation** that describes how the velocity changes over time as a function of acceleration. The index k represents the current discrete time index. The equivalent continuous time index can be derived by multiplying k by our time increment $\Delta t = h > 0$. From now on we will represent this finite difference equation with an '=' symbol because it converges to the continuous differential equation solution as Δt decreases to zero. Also, we'll systematically use Δt instead of h for the time increment. When dealing with ODEs, Δt is the more common notation.

Using this same method we can compute a finite difference equation that expresses how position changes

$$\frac{dp}{dt} = v \tag{3}$$

$$\frac{p(t+\Delta t) - p(t)}{\Delta t} \approx v(t) \implies p_{k+1} - p_k \approx v_k \Delta t \implies p_{k+1} = p_k + v_k \Delta t \tag{4}$$

For more information see Appendix C of the ROB 101 Booklet.

[1] "In an internal combustion engine, the throttle is a means of controlling an engine's power by regulating the amount of fuel and air entering the engine. In a motor vehicle the control used by the driver to regulate power is sometimes called the throttle, accelerator, or gas pedal."

2.2 What are States and What are Control Inputs?

We will now stack the position and velocity into a vector so that we can write the difference equations for the car in matrix form. You have come to expect that from us, right?

$$x_k := \begin{bmatrix} p_k \\ v_k \end{bmatrix}$$

As a point of vocabulary, the vector x_k, which combines the position and velocity of the car, is called the **state** of our model, while the force that we apply to the wheels of the car as a function of time is called a **control input** because we are allowed to vary it as we wish (we are in control of what value is applied at each instant of time). When you are driving, you vary the force through the accelerator pedal, the brake, and through the gear you have selected.

From (2) and (4), we can express the equations in matrix form

$$x_{k+1} = \begin{bmatrix} p_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} p_k + v_k \Delta t \\ v_k + a_k \Delta t \end{bmatrix} = \begin{bmatrix} p_k + v_k \Delta t \\ v_k + \frac{F_k}{\mathcal{M}} \Delta t \end{bmatrix} \tag{5}$$

Notice that the state vector at the next time instant x_{k+1} can be rewritten as a function of state vector at current time x_k and the wheel force F_k,

$$\begin{aligned} x_{k+1} &= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_k \\ v_k \end{bmatrix} + \begin{bmatrix} 0 \\ \Delta t / \mathcal{M} \end{bmatrix} F_k \\ &= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ \Delta t / \mathcal{M} \end{bmatrix} F_k \end{aligned} \tag{6}$$

F_k is the **control input** that is applied to the system at time t_k to drive current state x_k to its value at the next time instant, x_{k+1}. Equation (6) in control theory is referred to as the **discrete dynamics**. The discrete dynamics equation can be generalized to any system with finite difference equations. It is common practice to represent the matrix that multiplies the previous state as A and the matrix that multiplies the control input as B. The control input is also represented by the letter u.

Models of the form

$$x_{k+1} = Ax_k + Bu_k. \tag{7}$$

are called state-variable[2] models. We will use the notation shown in (7) for the remainder of this project. To be extra clear, in our case,

$$A := \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, B := \begin{bmatrix} 0 \\ \Delta t / \mathcal{M} \end{bmatrix}, x_k := \begin{bmatrix} p_k \\ v_k \end{bmatrix}, \text{ and } u_k := F_k.$$

2.3 Predicting the State Vector at Future Instances of Time

Now we have a way to compute the state at the next time instant based on the current state and current input! We can also iterate forward and **predict** that state at some future time, say N, as a function of a hypothesized input sequence, $\{u_0, u_1, \ldots, u_{N-1}\}$ as follows

$$\begin{aligned} x_0 &= \text{ given} \\ x_1 &= Ax_0 + Bu_0 \\ x_2 &= Ax_1 + Bu_1 = A(Ax_0 + Bu_0) + Bu_1 = A^2x_0 + ABu_0 + Bu_1 \\ x_3 &= Ax_2 + Bu_2 = A(A^2x_0 + ABu_0 + Bu_1) + Bu_2 = A^3x_0 + A^2Bu_0 + ABu_1 + Bu_2 \\ &\vdots \\ x_N &= A^Nx_0 + A^{N-1}Bu_0 + A^{N-2}Bu_1 + \cdots + ABu_{N-2} + Bu_{N-1} \end{aligned} \tag{8}$$

[2] You can learn about them in ME 360, EECS 367, EECS 460, and EECS 560.

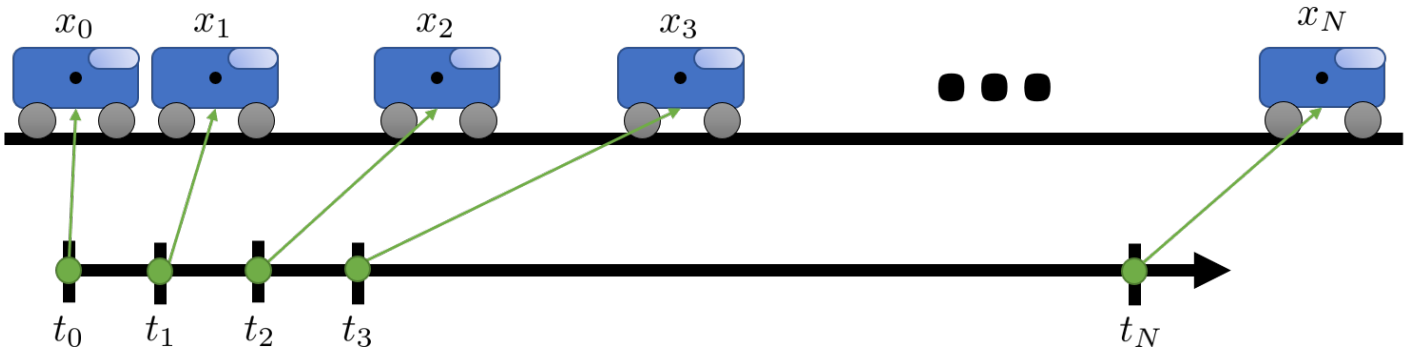


Figure 3: An illustration of a car moving along a line, with its state shown at various times. At each point in time, x_k provides the position and velocity of the car at time $t_k = k\Delta t$.

We now focus on that last line of (8). We will see that it is a gold mine. We note that

$$x_N = A^N x_0 + A^{N-1} \cdot Bu_0 + A^{N-2} \cdot Bu_1 + \cdots + A \cdot Bu_{N-2} + Bu_{N-1} \tag{9}$$

$$x_N = A^N x_0 + \begin{bmatrix} A^{N-1} \cdot B & A^{N-2} \cdot B & \cdots & A \cdot B & B \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} \tag{10}$$

For compactness of notation, we define

$$S := A^N \tag{11}$$

$$M := \begin{bmatrix} A^{N-1} \cdot B & A^{N-2} \cdot B & \cdots & A \cdot B & B \end{bmatrix}, \text{ and} \tag{12}$$

$$u_{\text{seq}} := \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-2} \\ u_{N-1}, \end{bmatrix} \tag{13}$$

which leads to our most important equation so far:

From x_0 to x_N as a function of the control sequence.

The following equation will be the basis of all of our work on steering the car around and balancing the Segway. If you do not understand it, you have better reach out NOW!

$$x_N = Sx_0 + Mu_{\text{seq}}. \tag{14}$$

What happens if you know the state at time k and want to predict it at time $k+N$? It works like this

$$x_{k+N} = Sx_k + Mu_{\text{seq}}, \tag{15}$$

where this time,

$$u_{\text{seq}} := \begin{bmatrix} u_k \\ u_{k+1} \\ \vdots \\ u_{k+N-2} \\ u_{k+N-1} \end{bmatrix}.$$

You will use this relationship at least 15 times in Project 3. Like we said, if you have any questions, now is the time to speak up!

2.4 Open-loop Driving via Optimization

For our car model, $x_k \in \mathbb{R}^2$. Let us look at (14) when $N = 4$, for example,

$$x_4 = A^4x_0 + \begin{bmatrix} A^3 \cdot B & A^2 \cdot B & A \cdot B & B \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix}.$$

We rewrite the last line as

$$x_4 - A^4x_0 = \begin{bmatrix} A^3 \cdot B & A^2 \cdot B & A \cdot B & B \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{16}$$

Major Observation: Suppose we know x_0 and we know where we want to be at time t_4, in other words, we know x_4. We seek to choose the input sequence u_{seq} so that the system moves from x_0 to our **goal state**, x_4. We observe that the left hand side of (16) is known to us while the right hand side is to be determined. The left hand side is a 2-vector while the right hand side depends on a 4-vector. Equation (16) is therefore an **underdetermined** system of linear equations in the variable $u_{\text{seq}} \in \mathbb{R}^4$.

Hence, a reasonable way to solve (16) for u_{seq} is to seek the control sequence of minimum squared norm such that (16) is satisfied. The general version is to seek the u_{seq} of minimum squared norm such that (15) is satisfied. This may be a good time to review Section 9.9 of our textbook! Look for the bright green box with the title **Minimum Norm Solution of Underdetermined Equations**.

Optimal Control Sequence Steering the System from x_k to x_{k+N}

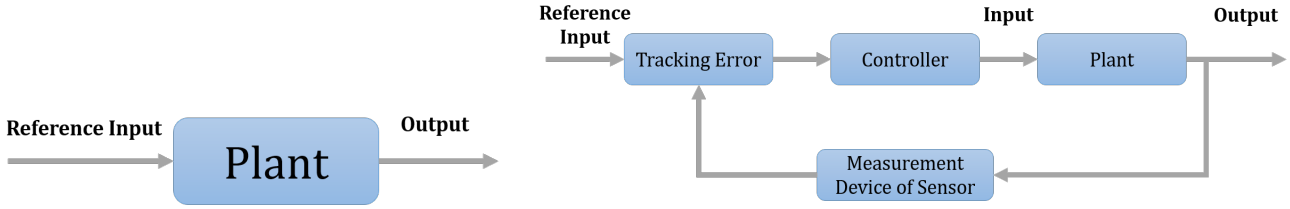
$$u^*_{\text{seq}} := \underset{x_{N+k} - S x_k = M u_{\text{seq}}}{\arg\min} ||u_{\text{seq}}||^2 \tag{17}$$

We note that

$$||u_{\text{seq}}||^2 = \sum_{k=0}^{N-1} (u_k)^2.$$

It is not too far fetched to interpret $(u_k)^2$ as the energy in u_k. If we do this, then the control sequence we have computed in (17) has the happy interpretation of being the control solution of minimum energy that drives our car from x_k to x_{k+N}. That is quite remarkable. **It is very useful for the Green Economy!**

You will use the least-squares problem in (17) dozens of times in Project 3. Please review the material before starting the project. In our textbook, the material is covered as solutions of underdetermined systems of linear equations.



(a) (b)

Figure 4: *Plant* is the generic term for the system to be controlled. A key difference between an open-loop control system shown in (a) and a closed-loop feedback system show in (b) is the absence of a loop in the former!

2.5 Closed-loop Control versus Open-loop Control and how this Relates to MPC

In open-loop control, or driving blindfolded, one looks ahead, sets a goal, computes an input sequence u_{seq} to drive one from their current position, let's call it x_0, to the goal, let's call it x_{Goal}, and that's the end of it. If nature (Murphy's Law) intervenes, too bad, you simply fail to reach your goal. This style of control is simple, but not very satisfying.

Our Take on Open-loop Control

Open-loop control, or driving blindfolded, works like this:

(a) remove blindfold;

(b) measure one's initial state, let's call it x_0;

(c) set a goal, let's it x_{Goal};

(d) set a planning horizon of N_{OL} time steps;

(e) solve $u^*_{\text{seq}} := \arg\min_{x_{\text{Goal}} - S x_0 = M u_{\text{seq}}} ||u_{\text{seq}}||^2$, where S and M are given in (11) and (12) with $N = N_{\text{OL}}$; and

(f) re-apply blindfold and hope for the best!

In closed-loop control, one constantly measures one's progress toward the goal and updates the input sequence u_{seq} as a function

of that information! Kind of makes sense. If someone else had not already invented the idea, you would surely have proposed it! It takes just a simple modification of our open-loop control process to transform it into a highly performing closed-loop controller methodology called Model Predictive Control or MPC for short.

MPC: Closed-loop Control via Least-squares Optimization

Model Predictive Control, or driving Eyes Wide Open, works as follows. It does not use a blindfold!

(a) Set a goal, x_{Goal}.

(b) Set a (relatively short-term) planning horizon for control of N_{ConHor} time steps;

(c) **FOR** $k = 1 : N$ **as long as you feel like driving**

- measure one's current state, x_k;
- solve

$$u^*_{\text{seq}} := \underset{x_{\text{Goal}} - Sx_0 = Mu_{\text{seq}}}{\arg\min} ||u_{\text{seq}}||^2,$$

  where S and M are given in (11) and (12) with $N = N_{\text{ConHor}}$;
- define $u_k := u_{\text{seq}}[1]$ and apply it to the actuators of your system (that is, the motors of your Segway or car).
- Either let physics do its thing and compute x_{k+1} for you, or, in case of a computer simulation, apply the control u_k to your mathematical model to compute $x_{k+1} = Ax_k + Bu_k$, and then wash, rinse, and repeat.

END

FAQ: What do I do with the remaining control values in u_{seq}? I just computed

$$u_{\text{seq}} := \begin{bmatrix} u_k \\ u_{k+1} \\ \vdots \\ u_{k+N_{\text{ConHor}}-2} \\ u_{k+N_{\text{ConHor}}-1} \end{bmatrix}$$

and you had me use only the first value! What do I do with the rest of them? Just throw them away?

Answer: Yes.

Q: But, but, but...

Answer: But what?

Q: This seems ridiculously wasteful!

Answer: Then put your blindfold back on and use them. Don't blame me, if you hit a tree!

Q: Can you at least tell me why I computed the extra control values, $\{u_{k+1}, \ldots, u_{k+N_{\text{ConHor}}-1}\}$

Answer: Of course. You needed to compute the full sequence so that you could predict the effort it would take to reach x_{Goal} and the relative urgency (necessity) of any corrective actions. You do this all the time when you ride a bike, a scooter, drive a car, or even walk across campus: you plan ahead and you keep updating your plan as you acquire new information. While you may not be

conscious of it, you are replacing your previously planned actions with the actions of your updated plan. So yeah, viewed in this manner, it's not so heretical to just throw away all but one of your planned actions! Welcome to the world of MPC!!

Q: How do I choose my control algorithm's planning horizon, N_{ConHor} ?

Answer: Carefully! Did you expect any other answer from us? OK, all kidding aside, if you want the ability to quickly and aggressively respond to disturbances, then a short horizon is necessary. If you want a smooth energy-efficient motion, then a longer horizon is important.

Remark: Do you recall how when you first learned to drive, you barely looked beyond the hood of the car? You were jerky on the steering wheel, throttle, and brakes? With more experience, you relaxed a bit and let your eyes focus farther down the road? That is what it means to have a longer control planning horizon, N_{ConHor}.

Q: Is there a lower limit on N_{ConHor} ?

Answer: Yes. Recall that the rows of the matrix M must be linearly independent. If you take the Linear Systems Course, EECS 560, you will learn that for a system with a single input, the shortest you can make the control planning horizon N_{ConHor} and have the rows of M be linearly independent is $N_{\text{ConHor}} = \dim(x)$, that is, the number of states in your model.

2.6 Model Predictive Control When xGoal does NOT Contain all of the States

Let's suppose that x_{Goal} only contains a few of the state variables. We'll do two cases,

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ while } xGoal = x_2^{\text{des}}$$

and

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_7 \end{bmatrix} \text{ while } xGoal = \begin{bmatrix} x_2^{\text{des}} \\ x_4^{\text{des}} \\ x_5^{\text{des}} \end{bmatrix},$$

where the superscript x_i^{des} means it is the desired value for x_i.

(a) Our suggested solution in both cases is to pose the problem as follows

$$u_{\text{seq}}^* := \underset{x_{\text{Goal}} - CSx_k = CMu_{\text{seq}}}{\arg\min} ||u_{\text{seq}}||^2,$$

where S and M are given in (11) and (12) with $N = N_{\text{ConHor}}$ and C is defined in terms of $xGoal$ as follows

(b) **Case 1:** Define $C = \begin{bmatrix} 0 & 1 \end{bmatrix}$ so that

$$Cx = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_2.$$

Because we know from (15) that

$$x_{k+N} = Sx_k + Mu_{\text{seq}},$$

it follows that

$$Cx_{k+N} = CSx_k + CMu_{\text{seq}}$$

and hence we can write our goal as

$$xGoal = CSx_k + CMu_{\text{seq}} \iff xGoal - CSx_k = CMu_{\text{seq}} \iff CMu_{\text{seq}} = xGoal - CSx_k.$$

(c) **Case 2:** Define $C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$ so that

$$Cx = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_4 \\ x_5 \end{bmatrix}.$$

Because we know from (15) that

$$x_{k+N} = Sx_k + Mu_{\text{seq}},$$

it follows that

$$Cx_{k+N} = CSx_k + CMu_{\text{seq}}$$

and hence we can write our goal as

$$xGoal = CSx_k + CMu_{\text{seq}} \iff xGoal - CSx_k = CMu_{\text{seq}} \iff CMu_{\text{seq}} = xGoal - CSx_k.$$

3 Segway Intro

3.1 How Segways Work

Figure 5: In this project, you will learn a method to keep a Segway upright! Left photo found here. Right photo found here

A Segway is a personal transportation vehicle that balances on two wheels. Invented by the American engineer Dean Kamen[3], Segways stirred up quite the buzz when they were brought to market in 2001.

One might be rightly perplexed by how such an odd-looking device manages to stay upright on its own–especially when a human is standing on it! Intuitively, one would deduce that a person who stands on the device will immediately fall forward flat on their face (or backward on their backs!) as the platform that the person stands on rotates in one direction or the other taking its human companion with it. If this is what you are thinking, do not fret because your intuition is exactly correct! If the Segway consisted of only two wheels and a platform to stand on, a human would very likely fall soon after trying to stand on the device.

[3] Yes, same person who set up FIRST Robotics. You can read more here.

But humans do not fall immediately after mounting a Segway–otherwise I am sure that we would see many lawsuits–and that is because Segways are not just two wheels and a platform. Segways also have motors that spin each of its two wheels individually, sensors that track the tilt, speed, and direction of the Segway, and microprocessors that use the information that the sensors give it to determine how fast each motor should spin to keep the Segway from falling.

Let me pose a question to you. If you were standing on top of a Segway and started to fall forward (thus causing the tilt sensors on the Segway to sense a forward tilt), which direction do you think the microprocessors will tell the wheels of the Segway to spin to maintain its balance: forward or backward?

If you answered forward, you would be correct! If you were to lean forward on a Segway, gravity would want to make you continue to fall forward, but the wheels of the Segway would spin forward faster to keep the Segway platform level beneath your feet. Thus, when you wish to move forward on a Segway, you lean your body forward to get the wheels on the Segway to spin forward. The more you lean, the faster the wheels will spin because ultimately the wheels are trying to spin fast enough such that the platform beneath your feet stays leveled. Similarly, if you were to lean backward on a Segway, the microprocessors would tell the wheels to spin faster backward to keep the platform beneath you.

This logic still applies when you wish to stand stationary on a Segway. When standing "still" on a Segway, you are really falling forward one moment and backward in the next (unless you have amazing balancing skills that allows you to easily balance on a platform supported by two wheels). Thus the wheels will also move forward one moment and backward the next to keep the platform level.

If you are still having trouble understanding how the logic of Segways works, it may help to draw a comparison with the human body. If you stood upright on solid ground and suddenly started to lean forward such that you became out of balance, you probably would not fall flat on your face. Your brain would know that you have lost balance because the fluid in your inner ear would shift in a manner such that it prompts your brain to tell your foot to step forward to prevent you from falling. If you continued to fall forward, your brain would have you put another foot forward, and another foot forward (causing you to walk–or run–forward), until it senses that you are no longer falling.

This is exactly what the Segway is doing, except instead of legs, it has wheels; instead of muscles, it has motors; instead of inner ear fluid, it has a set of sophisticated sensors; and instead of a brain, it has a series of microprocessors.

Here is a cool link which describes Segways in more detail if you are interested!

3.2 Typical Hardware on a Segway

Now that we understand how Segways work, let us take a look at the actual hardware components on a Segway. Segways can vary between manufacturers and product lines and so may have varied hardware components. However, there are certain components that must exist on every Segway in one form or another for the Segway to function appropriately. In our implementation, we will focus on these bare minimum components. These components are as follows:

- Sensors: position encoders, velocity sensors
- Microprocessor: controller
- Actuators: wheel motor
- Physical components: platform, wheel, shaft

The sensors track the position and velocities of the shaft angle and wheel angle. Note the singular "wheel" here. We will be working with one wheel because we will be using a simplified 2D model. The microprocessor–which will be the controller that we develop–will take the data from the sensors, interpret them, and provide torque outputs to the actuator. The actuator here is a single motor that powers the wheel. Finally, the physical components of our simple system (the platform, wheel, and shaft) will respond accordingly to the output of the actuator. The sensors pick up the new position and velocities, and the cycle continues until we have reached a desired state.

We provide a different take of Figure 4b here to further drive home the logic flow from sensors, to the microprocessor (where the feedback control law lives), to the motor(s), to the wheels, etc.

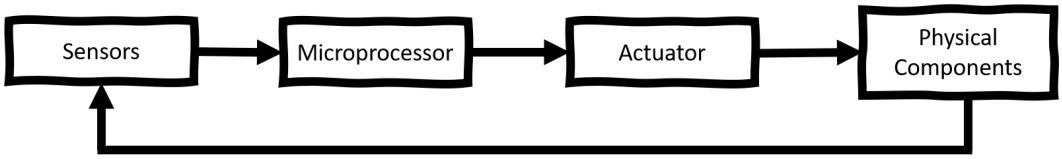


Figure 6: You may wish to refer to this diagram to help you visualize the logical flow of how the various components of our Segway interact with one another.

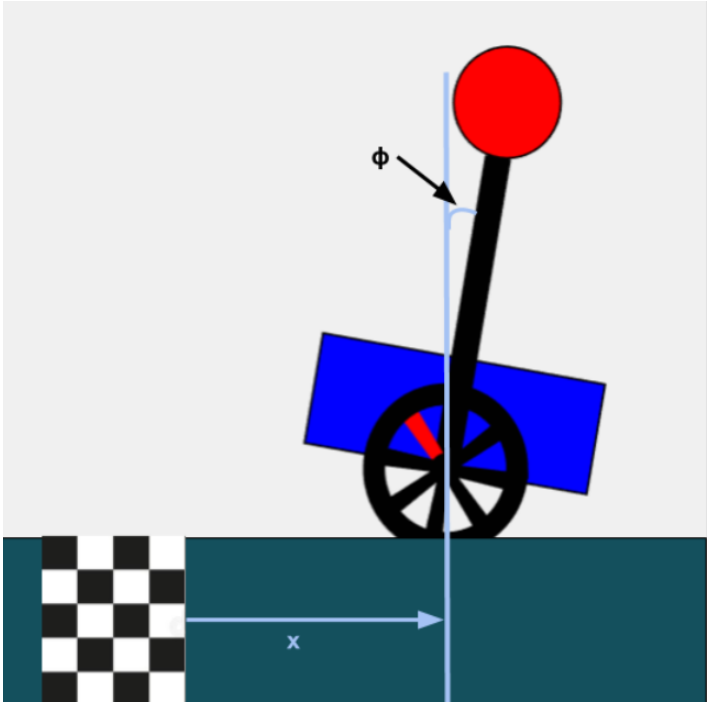


Figure 7: Segway state diagram

A mathematical model of our Segway will be given in the Julia portion of Project 3. You are now ready to start the project! Go!