Uprighting a MIP using MPC
Conrad Holda, Richard Lee
December 20, 2016

Abstract—In this report, the team explored the use of Dynamic Programming and Model Predictive Control (MPC) in order to have a mobile inverted pendulum (MIP) upright itself without any additional actuators. Due to the highly nonlinear dynamics of this system away from its equilibrium point, conventional controllers such as PID and LQR are unable to accomplish this task in isolation. However, by using MPC control, a trajectory can be planned over a fairly long time horizon and an optimal control sequence can be predetermined for all unstable states of the system. Additionally, the computation of an offboard optimal control policy ensures minimal use of onboard computing resources in the actual implementation of this controller. Finally, after uprighting the MIP through MPC, the system can switch to a more conventional controller such as LQR, in order to return to its initial horizontal position. The team was successfully able to generate this control policy and simulate an uprighting and recovering maneuver in MATLAB.

Future additions to this control approach would be to apply reference tracking and MPC on the closed loop LQR system, after the MIP has arrived at its stable equilibrium state. This would then allow the vehicle to follow more complex position trajectories. As a final step, the team would integrate this complete control policy on the physical system.

I. INTRODUCTION

The MIP is a classic controls problem taught to many UCSD undergraduates learning controls. At UCSD, students are presented with the task of balancing a MIP (pictured in Figure 1) starting from a near upright position. In order to do so, students are instructed to stabilize the system by using successive loop closures. An inside (faster) loop keeps the MIP upright while the outside (slower) loop keeps the MIP from wondering off in the horizontal direction.

The control theory behind MIPs is also applicable outside the classroom context, in the design of toys and modes of transportation. In recent news, Hoverboards have come to be popularized by Vine stars for their novelty and entertainment value. In more practical applications, this segway-like motion is appealing to the design of mobile, commercial robots for its minimalist use of actuators and general simplicity. However, in order to further promote the widespread adoption of autonomous segway-like robots, an upright maneuver will be required in the robot’s repertoire. This would avoid the need for human intervention in the case of an accident.

II. EQUATIONS OF MOTION

Equations of motion for a MIP were derived for a two dimensional model of the dynamic system. For the purposes of this project, the wheel was modeled as a uniform cylinder and the body modeled as uniform rod, however more complex geometries can also be modeled by altering the moment of inertia terms that parametrize these assumptions ($I_w$ and $I_b$ for the inertia of the wheels and rod, respectively). The complete derivations can be found in Thomas Bewley Ph.D. book, Numerical Renaissance [1]. The following two nonlinear equations are in terms of $\theta$, the angle between the body and the vertical position, and $\phi$, the angle between the starting orientation of the wheel and the current rotation of the wheel.

\begin{equation}
[I_w + (m_w + m_b)r^2]\ddot{\theta} + [(m_blr)\cos\theta]\dot{\theta} - (m_blr)\sin\theta(\dot{\theta})^2 = \tau
\end{equation}

\begin{equation}
[(m_brl)\cos\theta]\ddot{\theta} + (I_b + m_bl^2)\dot{\theta} - (m_bgl)\sin\theta = -\tau
\end{equation}

The variables used in the simulation were based on the UCSD’s MAE 143C course’s MIP. The mass of the wheels, $m_w$, and body, $m_b$, are 54$g$ and 263$g$, respectively. The inertia terms $I_w$ and $I_b$ are taken about the respective part’s center of mass, where the inertia of the wheels is 1.223 $\times$ $10^{-4}$ $kg \cdot m^2$ and the inertia of the wheels is 4 $\times$ $10^{-4}$ $kg \cdot m^2$. The length of the body is twice $l$ ($l$ is the length from the center of mass of the body to the wheels’ center); $l$ is 0.036$m$. The radius of the wheel, $r$ is 0.034$m$. Finally the gravity constant $g$ is taken to be 9.81$m/s^2$. $\tau$ is the torque applied by the motors which attach the wheels and body through a gearbox.

In MATLAB, these two equations (1) (2) were consolidated into one nonlinear equation to only be in terms of $\theta$ and its time derivatives, as well as the input $\tau$. $\phi$ is only important if horizontal position is desired; however, for the purposes of
uprighting the system, \( \phi \) is not used for reasons discussed in the next section. To discretize the system, the team used an Euler discretization of the continuous dynamics.

### III. MPC: Dynamic Programming

The team decided that to solve this problem, the Dynamic Programming approach would be necessary due to the nonlinear equations of motion. The equations could not be linearized about one point because the MIP would travel large angles to transition from laying flat to upright. A second approach the team considered was to design a trajectory for the MIP and linearize the equations at each time step. However, this would require a careful consideration of the path according to physical limitations of the MIP; therefore, this latter approach was not implemented.

There were quite a few variables the team had to tune in order for the MPC controller to be functional. The first of which was the gridding of the states and input. A finer grid would allow for the control policy to be more accurate at a particular state, but would also require more computing time to generate. A balance needed to be struck between designing a control policy that would allow us to stabilize the system and one that kept computation time to a minimum. The team finally decide to have the \( \theta \) and \( \dot{\theta} \) state gridded into 51 points each and the input \( \tau \) gridded into 26 points. \( \theta \) was constrained between \(-\pi/2\) and \(\pi/2\) as these are the angular limits of the MIP pictured in figure 1, which rests on its bumpers at these orientations. \( \dot{\theta} \) was limited between \(-5\) and \(5\) radians per second only for the purposes of generating a finite number of grid points. (However, this limit was tuned to ensure it was never limiting the system). Additionally, as stated in the previous section, the \( \phi \) and \( \dot{\phi} \) states were not taken into consideration because doing so would require the program to check for a multiple of \( n^2 \) more grid points, where \( n \) is the number of grid points in both \( \phi \) and \( \dot{\phi} \). This resulted in about a 100 second computing time for each step on a XPS 15 laptop equipped with a 6th generation i7 Intel processor.

The team also noticed while tuning the MPC controller, that the time step was important in stabilizing the system. If the time step was too large the system would progress too far and the actuator would not be able to catch the MIP from falling. To remedy this, a small time step of 0.005 seconds was chosen. The team limited the selection to time steps achievable by a BeagleBone Black; the microcontroller has a 1 GHz processor so a 0.005 second time step would be possible while still allowing the microcontroller to execute other processes.

A horizon of 15 steps was chosen for the MPC controller. This equates to 0.075 seconds of time from beginning to the end. In simulating the MIP, the MPC controller only ran the first step of this horizon. At each step, a new control input would be chosen according to the input policy and with regards to the system’s current state.

### IV. Transitioning to LQR

With Dynamic Programming, the team was forced to focus solely on control of the angular position of the body; our optimal input policy is unable to control over state variables such as \( \phi \).

In order to improve on our control system, the team decided to implement a policy by which the system is able to switch from the original MPC controller to a more conventional LQR controller. In implementing LQR, the team is able to take advantage of the linear approximations one can make when close to the equilibrium position. Additionally, LQR control presents easily tunable and intuitive parameters for this particular problem, such that state costs can be proportionately weighted according to the importance of each state and the strength of nonlinear assumptions made.

In developing the second controller, the team needed to tune parameters such as the threshold values beyond which the system would transition to LQR control. Ultimately, the team chose a threshold magnitude of \( \theta = 0.2\text{rad} \). Having such a simple control policy could be beneficial in potential paths forward, as explained in a later section.

### V. Results

Our system started at an initial state of \(-\pi/2\) for the angular orientation of its body, with an angular velocity \(0\text{rad/s}\). Figure 2 illustrates the optimal input applied at each time step, while Figure 3 illustrates how this system’s state evolves as that input is applied. The optimal control law is a combination of the two controllers mentioned beforehand. Initially it depends only on the state terms \( \theta \) and \( \dot{\theta} \). After about one second, the threshold for \( \theta \) is reached and the LQR controller is initiated, controlling all four states. The input shown in Figure 2 was ultimately applied and simulated on the full system, dictated by the nonlinear equations (1) and (2). It can be seen in the graph of \( \theta \) in Figure 3 that the system ventures out of the linear regime when the LQR controller is active. However, although this controller was designed for a linear model of the system, these results show that the it is still able to keep the system stable under a nonlinear simulation of its state (for this particular simulation).

The simulated robot successfully uprighted itself in approximately 5 second while traveling a total distance of approximately 3.5 meters before returning to its original horizontal position.
VI. Paths Forward

Having developed an control policy capable of balancing a MIP from a flat, stationary position and returning it to its initial horizontal position, the next step would be to apply reference tracking and MPC on the closed loop system generated by LQR control. Under the assumption that the wheels do not slip, developing a reference trajectory for the state $\phi$ would be equivalent to developing a trajectory for the horizontal position, $x = r\phi$. This trajectory would allow the MIP to be brought to different desired positions and provides the opportunity for even greater autonomy in segway-like robotic applications.

Finally, after developing such a control policy, the team’s ultimate goal would be to implement the policy on a physical MIP, such as the one found in UCSD’s MAE 143C course. Doing so would allow us to test issues such as neglecting the dynamics of the actuators or noise from the IMU sensor. Additionally, the Euler discretization estimates could be compared with the actual states of the physical system. Although the sampling time used for the purposes of simulation is very small (0.005 seconds), having the input implemented on the real-world system could provide additional validation.

VII. Conclusion

As a result of combining two individual control schemes, one generated by Dynamic Programming and the other generated by LQR, according to a policy through which they can be alternated, the team was able to develop a maneuver to upright a MIP and return it to its initial horizontal position. The implications of this controller is that it grants segway-like robotic vehicles greater autonomy. Without having to rely on human intervention in the case that a robot falls, the implementation of segway-like locomotion become a more attractive and feasible design decision for robotic manufacturers.