Good morning everyone. My project is the control of a rolling-balancing mechanical system: the disk-on-disk system. ||

Today we will be covering three things. The first up is a broad overview of the project, including an update since the video uploaded to YouTube. The second is a response to some of the questions posted to the progress video. Finally, I will be answering any questions from anyone here today.||

Study of the disk on disk system is motivated by two things: the fact it is an underactuated system, and the fact it isolates the rolling type primitive. Underactuated systems are systems which cannot be commanded to follow an arbitrary trajectory in a configuration space, which typically arises from there being less actuators than degrees of freedom in the system. Most of us here are probably familiar with the Segway robot, which is good example of an underactuated system.||

Primitives are the classifications of non-prehensile manipulation types, which is any manipulation that doesn’t involve the gripping or grasping action used by people to pick things up. Our tendency to use gripping actions to do everything has led to robotic manipulators that solely utilise the gripping action which constrains their kinematic workspace to a smaller region of influence than is theoretically achievable.||

The layout of the system used for modelling is shown on screen. The upper disk is called the object, and the lower disk is called the hand. The disk on disk system was modelled using the Euler-Lagrange method in the pursuit of torque-based control using the Linear Quadratic Regulator. This model was later used to develop the Hamiltonian model to enable velocity-based control, which will be discussed later.||

Control was achieved in simulation, with balancing achieved and satisfactory reference tracking. The results of the simulation, shown here, are used to design the experiment apparatus. The main constraints are the motor torque and velocity requirements. The simulation shown on screen captures most typical behaviour expected during a balancing action, with the required torque and velocity values being well within the limits of a hobbyist DC motor. ||

Here we can see the 3D CAD model on the left and the final product in the right two images. The disks are made from clear polycarbonate to reduce weight and cost, whilst the frame is made from 20mm aluminium square hollow section. The enclosure plates are also made from clear polycarbonate as they must be transparent for the computer vision measurement system. The motor and electronics are housed under the U channel section which runs the length of the frame in the centre. The blue dots in the right image are markers for computer vision feature detection.||

State estimation for control is achieved using different methods for the two disks. The lower disk can utilise a rotary encoder attached to the motor shaft, allowing estimates of position and velocity. For the upper disk it is not feasible to use measurement devices that require contact with the object, so a feature recognition computer vision process is developed. ||

Feature recognition is carried out using a saturation thresholding process. On the left is the test image in its HSV version where we can see the blue circles are well defined from their surroundings. By extracting the saturation channel, and thresholding the image based on the mean saturation value, we are left with a binary representation of the image, shown here on the right. ||

This binary representation is suitable for use with a blob detection algorithm. By comparing the value of a pixel to its surroundings, regions of connected pixels can be determined, called blobs. The centroid of the blobs is then used to calculate an angle offset of the upper disk. In this photo, the top disk has a marker attached, whilst the lower blue circle is stationary and used as a reference. This process is implemented using two different cameras, which I will go into later while answering some of the questions posted online.||

System identification was carried out on the DC motor, which allows for any non-ideal behaviour in the physical system to be modelled and accounted for, enabling precise control over the motors behaviour. With the standard DC motor model shown in the upper picture, we expect a sinusoidal input voltage to result in sinusoidal output velocity and current, which is clearly not the case, as shown in the lower plot. There is a phase offset between the velocity and voltage, along with a significant deadzone. ||

Through the system identification process, it was identified this behaviour is a result of several different types of friction in the motors gearbox, and a friction profile was developed. The model combines Stribeck, coulomb, viscous and static friction models, with a plot of the friction profile versus motor shaft velocity shown on screen. Near zero velocity, the friction coefficient peaks to account for the deadzone behaviour, and transitions to a viscous and Coulomb model far from zero.||

Using the developed friction model, the system is simulated to determine the difference between predicted and measured outputs for a given input. As we can see, the agreement between velocity plots is good, and the agreement between current plots is relatively poor, which led to extremely poor performance when using torque-based control. ||

On the left is a plot of resultant velocity for a demanded sinusoidal torque input using torque control, and on the right is a plot of demanded and measured velocity for a sinusoidal velocity input using velocity control. In this instance, good performance is indicated by the output velocity matching the shape of the input signal, which is a sine wave. Clearly, the velocity control method is far superior to torque control in achieving this goal. It is also important to note that the velocity control method does not require any form of system identification to be carried out and simply uses PID for control allocation. The velocity control method does require a new plant, controller, and observer model to be developed to solve a causality issue, but this is easily achieved using the already derived Lagrangian model. Implementing velocity control solves the performance issue found when using torque control.||

Putting this all together brings us to the results, which are unfortunately not very exciting. Balancing has not been achieved in experiments due to issues with the image acquisition process which forces the achievable control frequency to be below the lower limit determined in simulation.

This brings us up to date and to the first YouTube question.||

The answer to “were there problems arising from the image acquisition delay” is yes, the delay made it impossible to achieve balancing.

The minimum delay consistently achievable in the experiment was 80 milliseconds, which dictates the control and measurement frequency. By aligning the delay time with the timestep of the system, the Kalman filter predicts one delay period ahead, ensuring the predicted and measured outputs can be correctly compared for the updating of the Kalman filter. If the Kalman filter is run with a smaller timestep than the delay period, the predicted and measured outputs will never align, causing the Kalman filter to not function correctly. ||

Slowing the Kalman filter, and in turn the control frequency, to match the delay period is where the issue arises. At 10 hertz the system shows signs of instability and becomes unstable at 9 hertz. Typically, a control frequency several times higher than the minimum is implemented to account for any other non-ideal behaviour in the physical implementation, and the 12 hertz control frequency achievable does not allow enough overhead in this instance. ||

The next question relates to the cause of the delay in the video stream.||

The first of the two cameras tested was a Microsoft webcam which is ironically no longer compatible with windows operating systems. To get it working, third party software drivers were used, and it was also piped through a second piece of software that acts as a virtual webcam. The multiple layers of software to make this camera work were assumed to be the source of the delay which led to the use of the second camera.  
The raspberry pi camera board is capable of 90 frames per second and can act as a webcam for a computer using the UVC protocol, which is a standard protocol for USB cameras. When using a camera for a video stream, there is a level of unavoidable delay in the video stream due to the way most cameras work. ||

The camera produces an image using a rolling shutter, which captures an image by scanning from one side to the other and recording the image in rows, which takes some amount of time. The image is then post processed, converted from raw data to the correct output format, and then transmitted. For a camera running at 30fps this entire image capture process takes a maximum of 33 milliseconds as the camera does not have an inbuilt memory buffer to store frames. The cause of the additional delay appears to be the combination of a memory buffer in MATLAB and the complications introduced by simultaneously streaming and capturing data on the raspberry pi. ||

The final question on the YouTube video was why I choose to use LQR over MPC. LQR was intended to be used initially as it is simpler to implement and debug, but the camera issues that arose stopped me progressing very far into MPC implementation. It has been implemented but is not performing well due to the measurement delay. ||

Thank you to everyone for your time. If anyone has any further questions, please feel free to ask them now.