PREREQUSITES:

- A robot, mounted with the effector pointing towards the device under test
- Arduino or compatible microcontroller.
- A device to test.
- A computer with 2 USB ports
- (optional, but highly recommended) an external USB camera.

SETUP:

- 1. Ensure robot is correctly assembled and positioned over the calibration card/device
- 2. Ensure the camera is positioned such that it can see the calibration card/device
- 3. Flash arduino with the appropriate ik.ino file from the R2B2 repository
- 4. Run smartcontroller.py -x
  - *a.* Follow the instructions to align the robot's movement axes with the orientation on the device. You may need to swap control lines between servos to make the X and Y axes align properly (+x should be to the right of y+)

BASIC OPERATION:

- 1. Run smartcontroller.py
- 2. The first step is to select your camera's orientation relative to the device so that the picture is "right-side-up" in the viewing screen. The result should be that the top of the screen of the device is at the top of the window displaying the camera's view.
- 3. The next step is to calibrate the perspective shift of the camera display. If using the calibration card, this may work automatically (if it puts green circles around just the four calibration card dots). Otherwise, use the mouse to click four points at the four corners of the device or four dots of the calibration card.
- 4. This should produce an image where the device/card appears as if viewed from above.

- 5. You will now be asked to select a region to check for changes, this helps the program ignore changes in the environment only pick up changes in the device's screen that would indicate an unlock. Click and drag a region on the window to select it as the region of interest. This region should not be too small, but should avoid areas the robot may be moving in if possible. Near the top of the screen is usually the best location.
- 6. The last step before button calibration is to set the "common Z" of the device. This is the height to which the robot will automatically move when calibrating the buttons, so it should be close to the device's screen but is not the final position so it does not need to be exact or even touching the device screen.
- 7. Button calibration is done with two input methods.
  - a. You can click on the screen and the robot will move to corresponding x/y coordinates. Depending on how obtuse an angle the camera is at the movements may or may not be very accurate. The closer to overhead the camera is positioned the more accurate using the screen to position the robot will be.
  - b. Use w,s to move in y-axis, a,d to move in x-axis, and q,e to move in z-axis. The amount each key-press moves the robot by can be adjusted by typing a number (be careful using large step-sizes when doing z adjustment as it can cause the robot to lift itself up, potentially affecting the alignment or even damaging the robot).
- 8. When the robot is positioned to be pressing a button, you can set that button's position by typing "Set" followed by the name of the button ("1", "2", "OK", etc.) without a space between "Set" and the name.
  - a. To check a button's position you can type "Goto" and the name of a previously defined button to move the robot to the defined location.
  - b. To save your configuration you can type "Write" and the name of the file to save to.
  - c. If you already had a configuration you can type "Read" and the name of the file to read from.
  - d. When all buttons have been configured simply press the Escape key to begin entering pins.
- 9. Each time the camera detects enough of a change that it may be

an unlock, it saves the image of the change in the pin-images folder, named with the name of the PIN it had just entered before capturing that image. Due to the unlock delay on some screens this may actually be one or two later in the list than the correct PIN.

COMMAND-LINE ARGUMENTS:

- -a, –android: Automatically adds an action to deal with the default Android device lockout of 30 second wait every 5 attempts and requiring the user to press the "ok" button in the popup dialog
- -d, –detectionthreshold ¡number¿: Change the default threshold for how different the camera's view must be to be considered "changed". If it is not detecting unlocks lower this, if it is detecting too many false positives raise this. Default is 2,000,000
- -f, –pinfile ¡file¿: reads the PINs to try, in order, from a file provided by the user
- -k, –keyconfig: runs camera-less; buttons are configured using the keyboard only and changes to the device screen are not detected
- -l, –loadpositions ¡file¿: loads the positions of buttons from a previously saved configuration specified by the argument. If this and –nodetect are provided then the entire visual configuration section of calibration is skipped
- -n, –nodetect: does not attempt to detect changes to the screen that would indicate an unlock. If this and –loadpositions are both provided then all visual configuration is skipped
- -p, –pattern: operate in pattern lock mode, so buttons are dragged between instead of pressed
- -s, –serialdevice ¡port name¿: the name of the serial port to connect to (windows typically COM#, Linux and OSX are usually /dev/tty* devices)
- -t, –maxtries ¡number¿: only tries up to maxtries number of PINs
- -v, –videonum ¡number¿: the slot the webcam occupies (typically 0 or 1)
- -x, –axis: run an axis-alignment check to start with to allow the user to match up the vertical and horizontal axes of the robot and the device
- -z, –reversez: inverts the sign of all Z coordinates on the arduino in case your servos turn the opposite way of normal. (Positive Z should

take the head up away from the ground and negative Z should take it towards the ground)

COMMON PROBLEMS

- *P:* On the very first run of the program, I get a serial error or timeout. *S:* You need to select the serial port your Arduino's USB virtual serial port is connected to on your computer. If you're unsure, see what serial port is reported by the Arduino programming software. Use the –serialdevice parameter to set the correct serial port.
- *P:* My camera shows a picture of me instead of the robot. *S:* You need a camera pointing at the device you're testing, and the shot shouldn't include much else that moves around (such as the robot's effector). We found an external camera to be the most effective. If your external camera is connected, but smartcontroller isn't using the correct camera, try the –videonum command line parameter.
- *P:* My robot is touching the button but the button is not being pressed. *S:* The stylus tip may not be properly grounded. Just grounding to one of the metal rods on the robot will not work. Connect the stylus tip to a ground pin on the Arduino for best results.
- *P:* My device shifts over time and so I cannot bruteforce it. *S:* Multiple solutions:
  - Strap your device in. Glue, tape, or zip-ties can all work to secure it in place. For our demos, we used pegboard and zipties for a solid, but non-permanent solution.
  - Lighter touch. Reconfigure your buttons to have the highest possible Z coordinate while still triggering the button.
  - Strap your robot in. Securing the robot will help prevent it shifting during operation and gradually accumulating error until it misses buttons.
- *P:* My robot punches the ground at the start of the program. *S:* Your servos probably rotate the opposite way of "normal". Pass the "-z" argument to compensate.
- *P:* My program detects locked screens too much/doesn't detect the unlocked screen *S:* Try adjusting the threshold by passing different values to the "-d" argument. Also try changing the "region of interest" selected in calibration. If it is too small it may be less likely to detect the unlock, if it is too large it might detect too many false positives. Move-

ment around the device during testing can also disrupt the detection, as even changes in shadows can result in a false positive.