Design And Validation of a Low Cost Dynamic Motion Simulation Tool

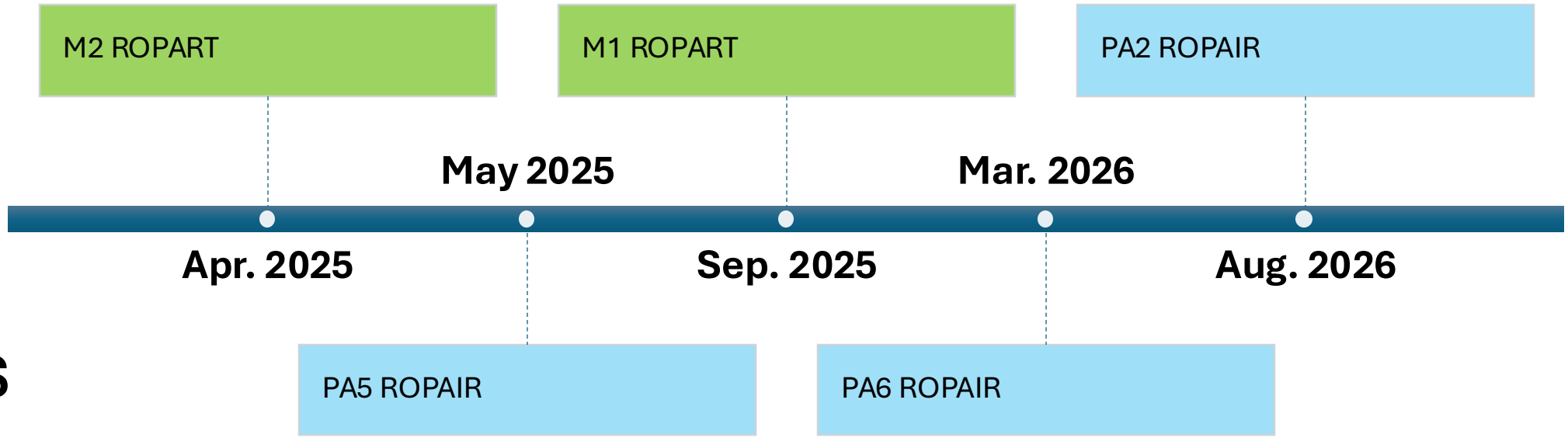
Chris Noble[2] Lucy Sim[1] Emma Horgan[2] Sau Fan Liu[2]

Emily Murphy[1] Deborah McFarlane[1] Scott Jones[1]

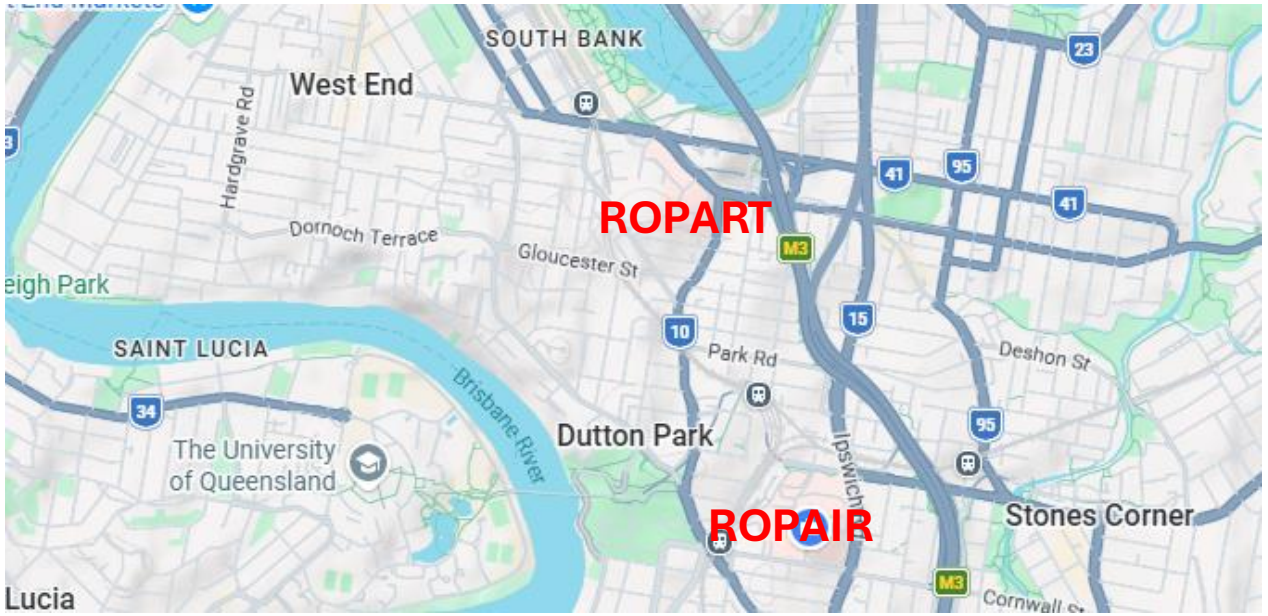
[1]ROPART and [2]ROPAIR

Metro South Hospital and Health Services

Queensland Health

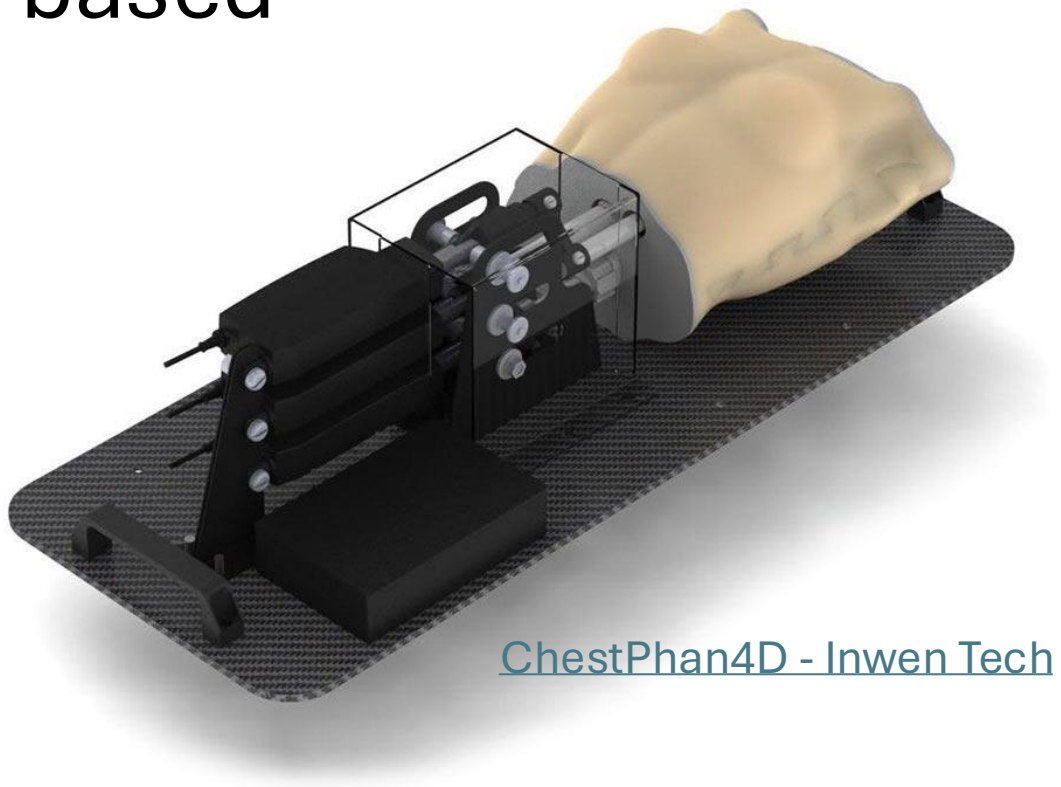


Princess Alexandra Hospital 5 AlignRT Installs 2025-2026



Why do we need a low-cost dynamic phantom?

- Commercial dynamic phantoms are costly
- SGRT systems are workflow based
- Needed for Commissioning
- Needed for Training

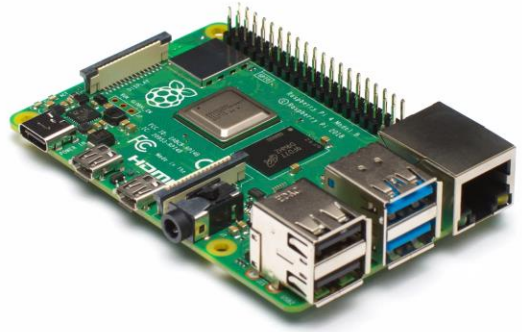


ChestPhan4D - Inwen Tech

Design Criteria

- Low Cost with easily obtained parts
- Easy to construct and repair
- Accurate and reproducible motion
- Good surface for SGRT
- Easy to modify and extend
- Easy and intuitive to use

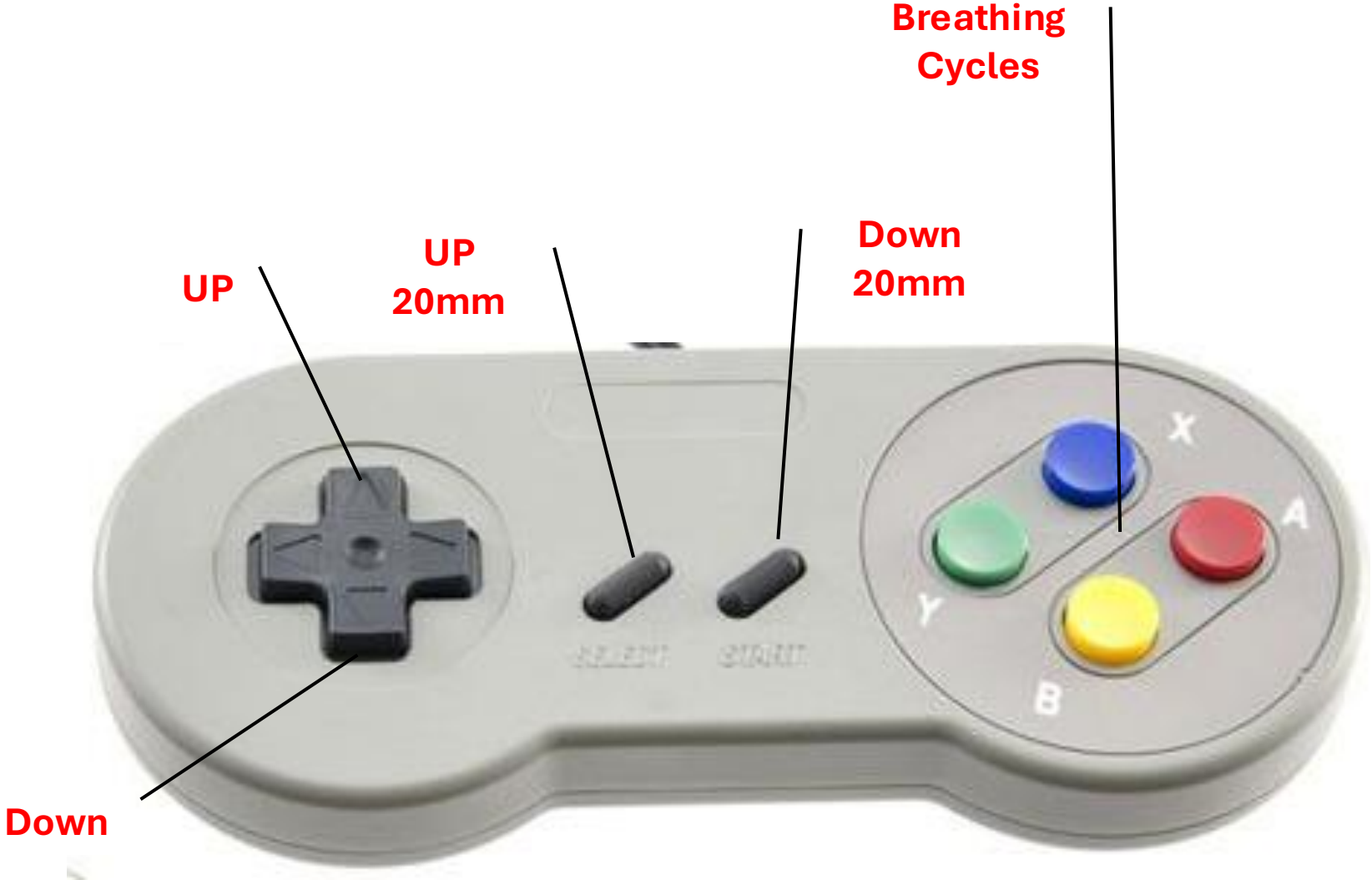
Components



Item	Price
Raspberry Pi 4B	$ 128.44
NEMA17 stepper motor with Lead Screw and Brass Nut	$ 68.26
12V DC 1.5A Slim Power Supply	$ 24.95
Mannequin	$ 17.89
24V/12V to 5V step down converter	$ 13.98
USB Game Controller (SNES Style)	$ 5.20
A4988 Stepper Motor Driver Control Board	$ 4.95
A4988 Stepper Motor Driver	$ 3.95
Total	$ 267.62

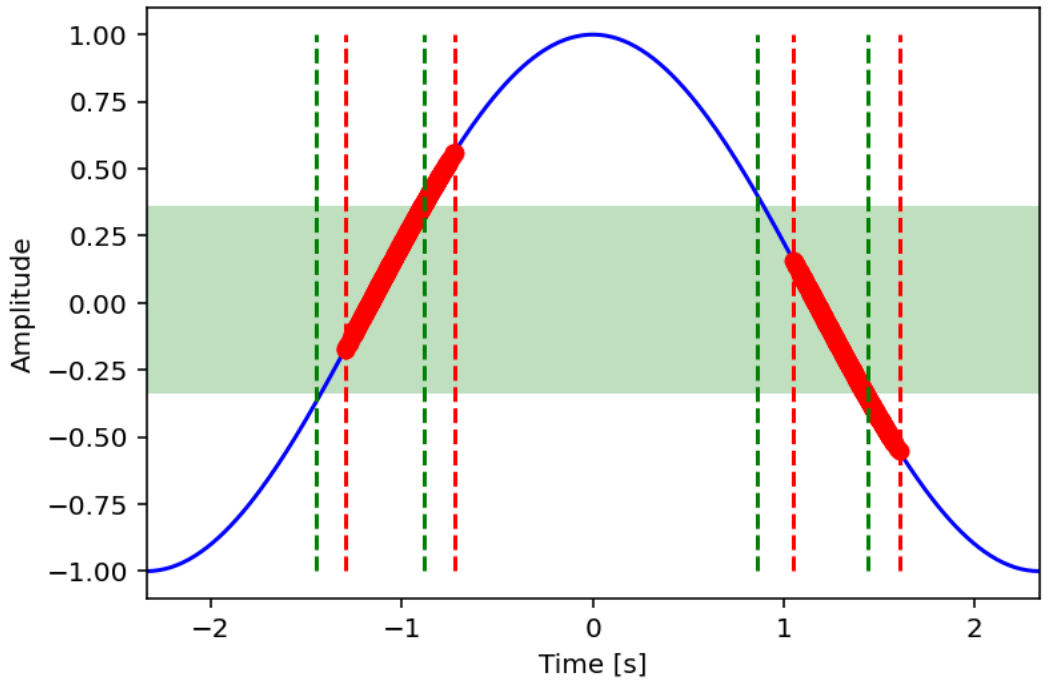
Code

- Raspbian operating system
- PyGame Library
- Run as a service
- Can be distributed on an SD card
- Easily reprogrammed



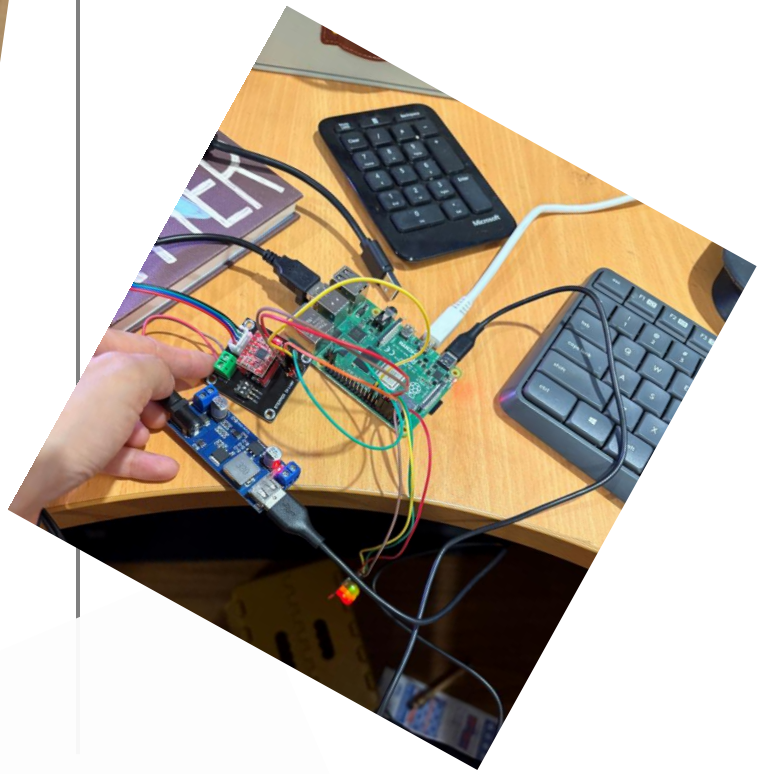
Initial Validation

- Accuracy of programmed amplitude
- 50 steps = 1mm
- Repeatability of motion
- Stability over time
- **Result:**
  Consistent, reproducible motion suitable for QA-level validation.



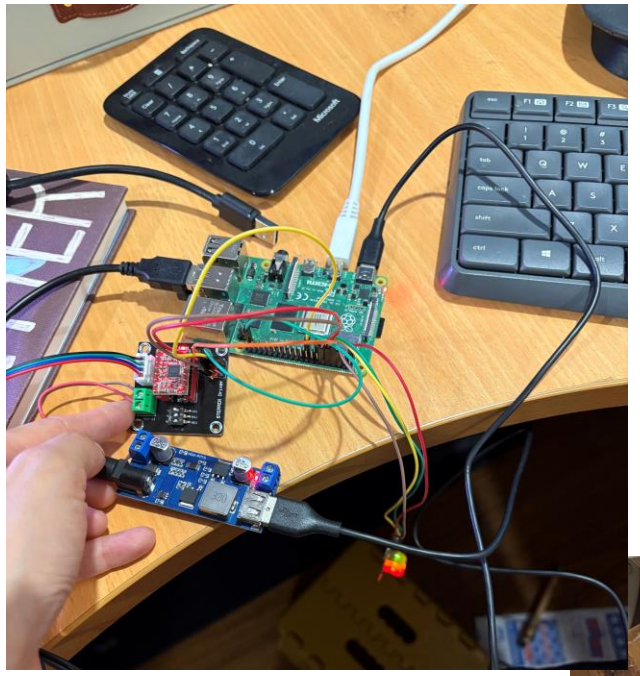

def motor_gpt(clockwise, numsteps,stepwidth, initdelay):
GPIO.output(stepper, GPIO.HIGH)
GPIO.output(green, GPIO.LOW)
stepwidth = 0.001
def motor_gpt(direction, numsteps):
GPIO.output(direction)
time.sleep(initdelay)
for i in range(numsteps):
GPIO.output(step, GPIO.HIGH)
time.sleep(stepwidth)
GPIO.output(step, GPIO.LOW)
time.sleep(stepwidth)
y = np.linspace(-1,1,50*amplitude_green)
print(50*amplitude_green)
steps_green = np.gradient(np.arcsin(y))
steps_green = steps_green*1/(np.sum(steps_green))
y_y = np.linspace(-1,1,50*amplitude_yellow)
print(50*amplitude_yellow)
steps_y = np.gradient(np.arcsin(y_y))
steps_y = steps_y*1/(np.sum(steps_y))
y_b = np.linspace(-1,1,50*amplitude_blue)
print(50*amplitude_blue)
steps_b = np.gradient(np.arcsin(y_b))
steps_b = steps_b*1/(np.sum(steps_b))
def motor_array(clockwise, steps, stepwidth):
GPIO.output(direction, clockwise)
t0 = time.time()
time.sleep(0.005)
for s in steps:
GPIO.output(step, GPIO.HIGH)
time.sleep(s)
GPIO.output(step, GPIO.LOW)
time.sleep(s)
t1 = time.time()
print(t1-t0)
x = 20
breathe_green = False
#pygame.init()
#pygame.exit.quit()
pygame.display.init()
pygame.joystick.init()
connected = pygame.joystick.get_count()
if not connected:
print("connect joystick")
while (connected < 1):
pygame.joystick.quit()
pygame.joystick.init()
connected = pygame.joystick.get_count()

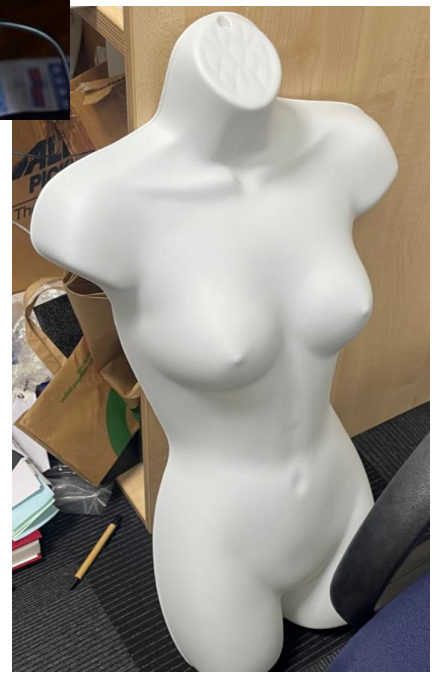

UP
DOWN
20mm UP
20mm DOWN
BREATHE
5mm
15mm
3mm
SKIP



Clinician Feedback

- We presented the prototype to:
  - Radiation oncologists
  - Medical physicists
  - Radiation therapists
- **Feedback Highlights:**
  - “Breathing amplitudes too small/large?”
  - “Need realistic variability”
  - “Include faster breathing patterns”
  - “Simulate breath-hold”





```
GPIO.output(amber, GPIO.HIGH)
GPIO.output(green, GPIO.LOW)

stepwidth = 0.001

def motor_go(clockwise, numsteps,stepwidth, initdelay):
    GPIO.output(direction, clockwise)
    time.sleep(initdelay)
    for i in range(numsteps):
        GPIO.output(step, GPIO.HIGH)
        time.sleep(stepwidth)
        GPIO.output(step, GPIO.LOW)
        time.sleep(stepwidth)

y = np.linspace(-1,1,50*amplitude_green)
print(50*amplitude_green)
steps_green = np.gradient(np.arcsin(y))
steps_green = steps_green*1/(np.sum(steps_green))

y_y = np.linspace(-1,1,50*amplitude_yellow)
print(50*amplitude_yellow)
steps_y = np.gradient(np.arcsin(y_y))
steps_y = steps_y*1/(np.sum(steps_y))


y_b = np.linspace(-1,1,50*amplitude_blue)
print(50*amplitude_blue)
steps_b = np.gradient(np.arcsin(y_b))
steps_b = steps_b*1/(np.sum(steps_b))

def motor_array(clockwise, steps, stepwidth):
    GPIO.output(direction, clockwise)
    t0 = time.time()
    time.sleep(0.005)
    for s in steps:
        GPIO.output(step, GPIO.HIGH)
        time.sleep(s)
        GPIO.output(step, GPIO.LOW)
        time.sleep(s)
    t1 = time.time()
    print(t1-t0)


#
x = 20
breathe_green = False


#pygame.init()
#pygame.mixer.quit()
pygame.display.init()
pygame.joystick.init()
connected = pygame.joystick.get_count()

if not connected:
    print("connect joystick")

while (connected < 1):
    pygame.joystick.quit()
```

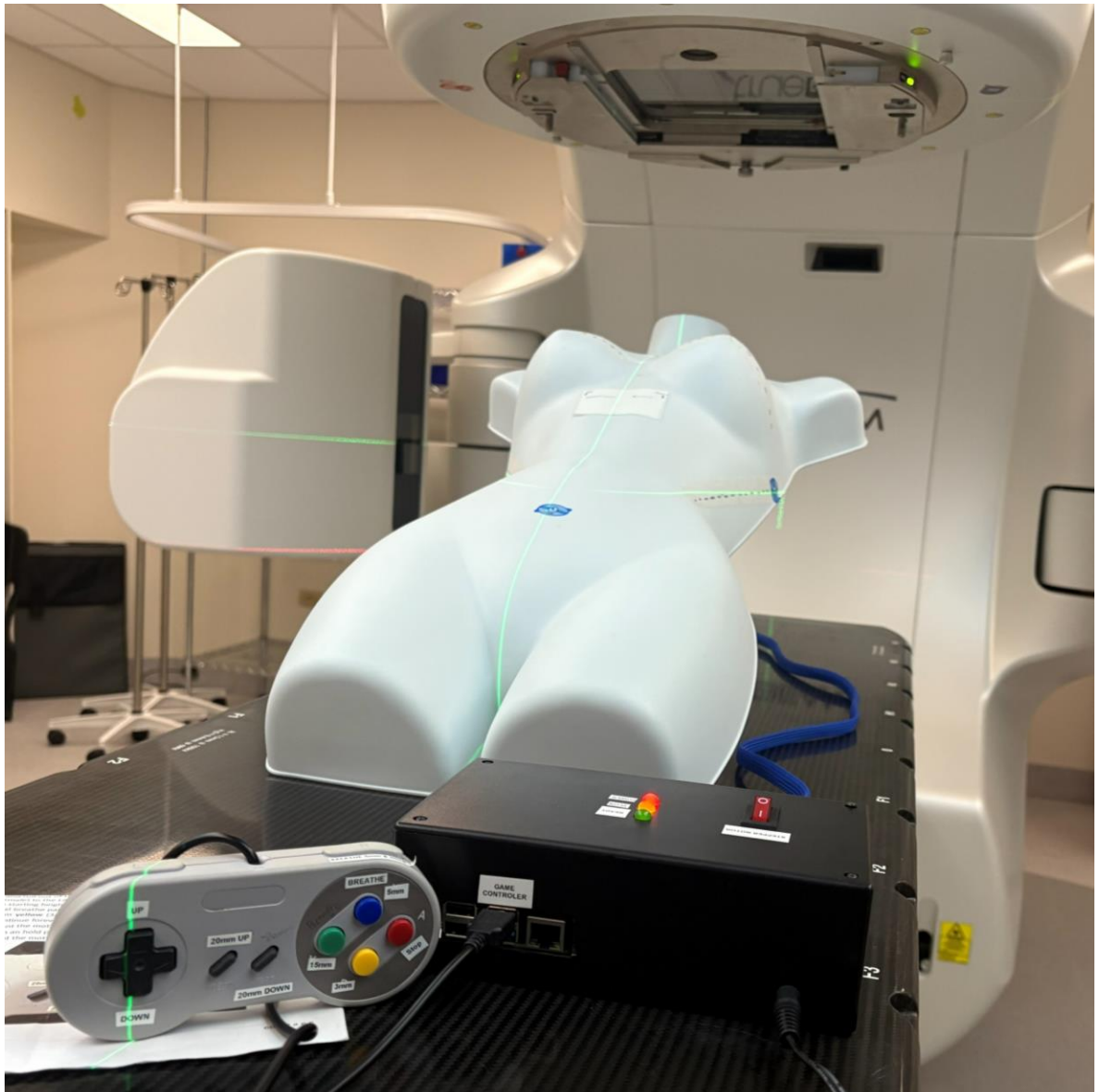
Results

- We implemented:
  - Variable amplitudes (e.g., shallow vs deep breathing)
  - Adjustable breathing rates
  - Irregular breathing patterns
  - Breath-hold simulation

Now capable of mimicking:

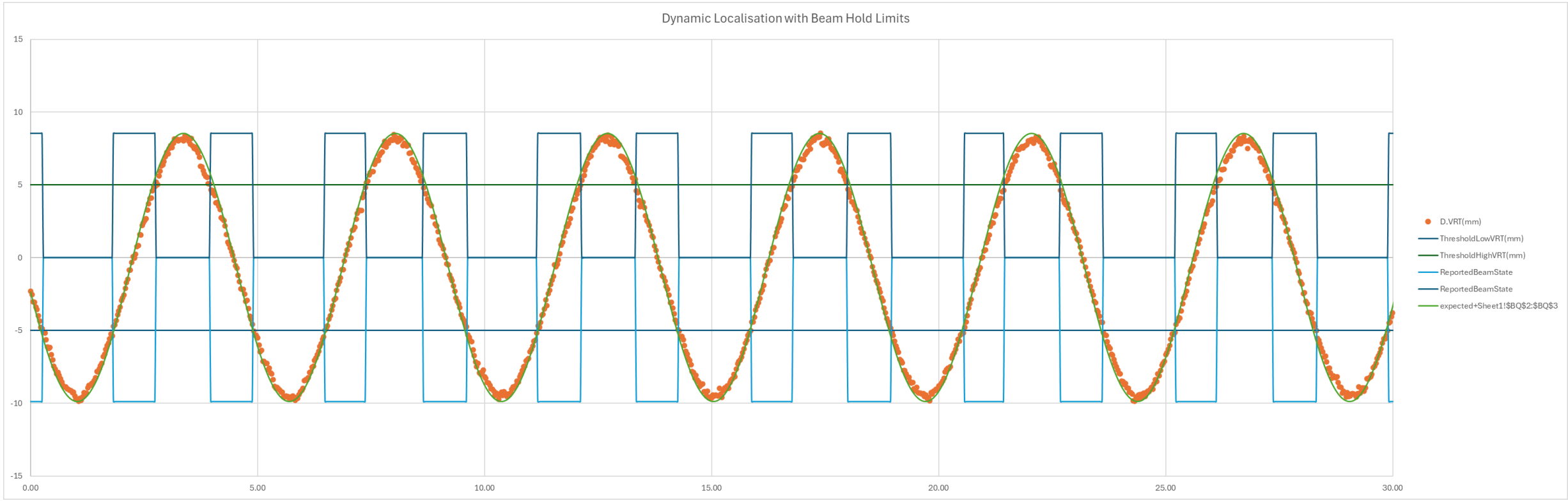
- Calm breathing
- Anxious/rapid breathing
- DIBH-style breath holds

Demonstration

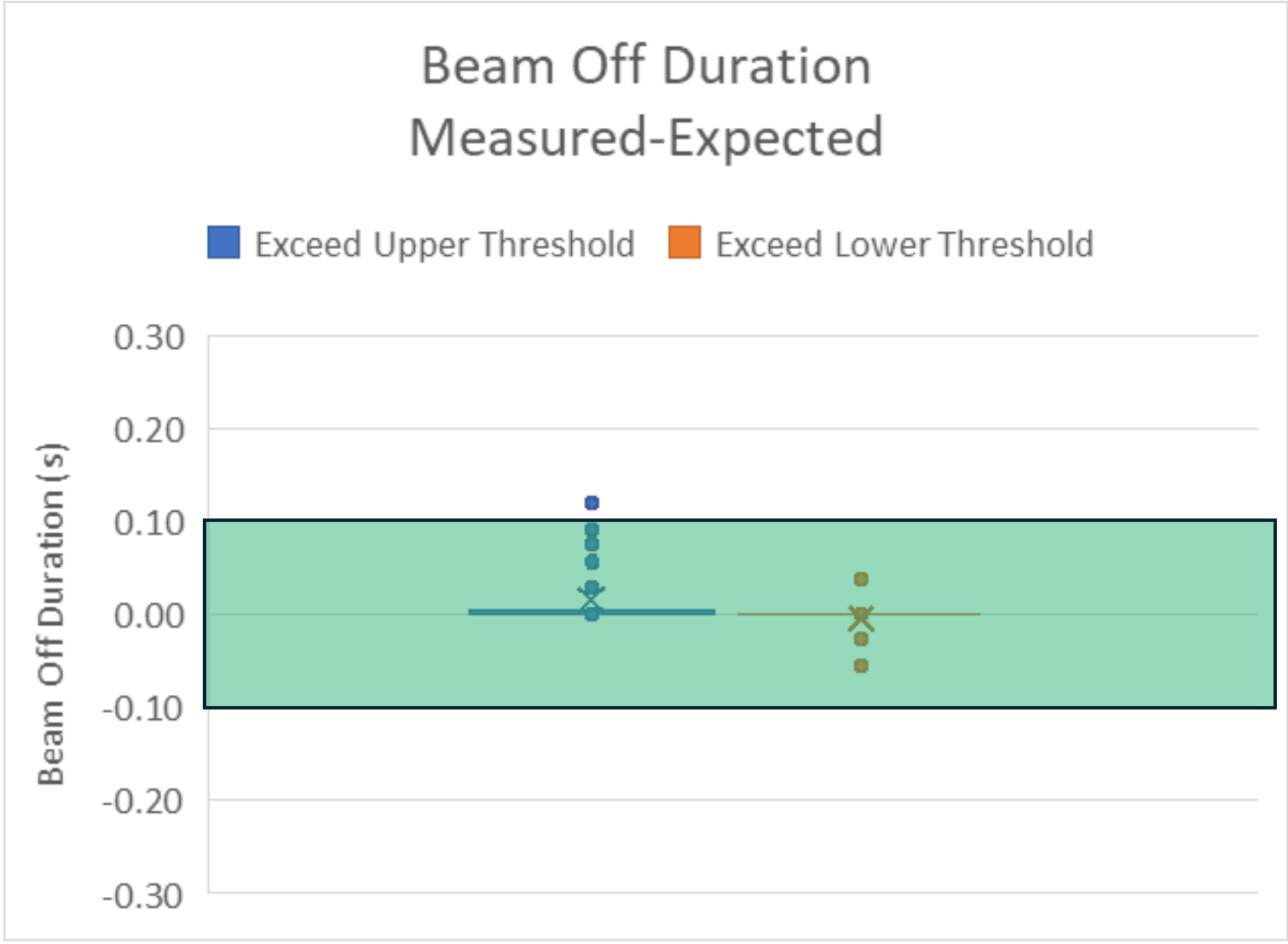


Dynamic Localisation

- Physics Commissioning Test
  - Option 1- programmable moving platform
  - Option 2- Christina Dynamic



Expected Beam Off vs Actual Beam Off



Beam gating and dosimetry

Beam	BPM	Period (s per breath)	Amplitude	Gating Window	% Diff
6 X	8	7.5	15	8	-0.05
6 FFF	8	7.5	15	8	-0.02
10FFF	8	7.5	15	10	0.13

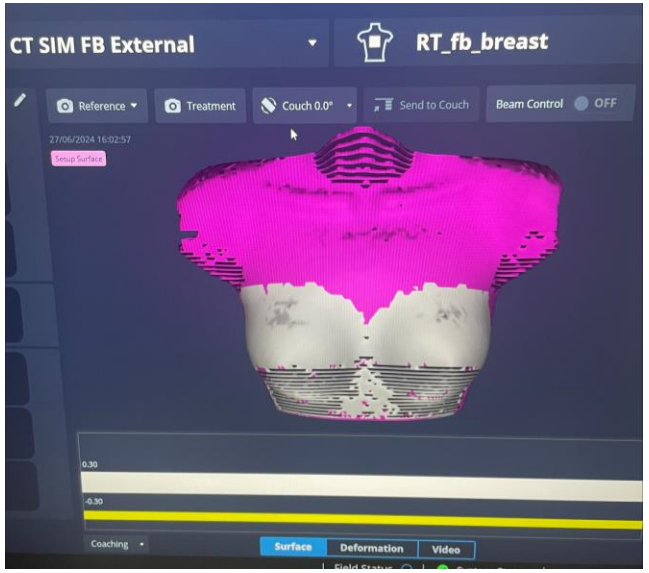
Conclusion

- Commissioning SGRT:
  - Effect of amplitude on tracking
  - Trigger Gating Beam On and Beam Off when outside tolerance window
  - Gating as patients fall out of breath-hold
  - AlignRT had 6 DOF gating capability, enabled testing in multiple gating axes

Unexpected Outcome: Training Tool

The phantom became valuable for RT education:

- Safe environment to:
  - Practice SGRT setup
  - Learn gating thresholds
  - Recognize motion artifacts
- Feedback from RTs:
  - Increased confidence before patient use
  - Better understanding of SGRT tolerances



Future Application

Potential evolution of Christina

- Add multi-directional motion
- Deformable?
- Incorporate patient-derived breathing traces
- Irregular breathing patterns
- Breath hold with drifts
- Share open-source build guide