**Introduction**  
This research focuses on creating a new input device using accelerometers wired to a wooden mannequin to manipulate virtual objects within the Unity game engine. By moving the physical arm of the mannequin, the arm of a character will bend the same way in the game world. We hope that the finished product could even be used in conjunction with virtual reality.

**Hardware and Software**  
- Arduino Uno Microcontroller  
- Up to fourteen MPU-6050s, a Gyro and Accelerometer by InvenSense  
- Unity Game Engine – Code in C#  
- Arduino IDE  
- Arduino communication software derived from Jeff Rowberg's I2Cdev library

**Basic Procedure**  
- Gyroscopes are attached to different limbs on the mannequin model assigned to each sensor  
- Data from the sensors is sent to the board via the I2C bus protocol, a common standard of peripherals information transfer  
- Data is then sent over the serial port to a desktop computer to calculate the individual rotation of each sensor and make calibration adjustments  
- Motion of each sensor is translated to a similar mannequin 3D model running on the Unity game engine

**From the Sensors to the Board**  
**Board Setup and Addresses**  
Because the I2C bus from the accelerometers to the Arduino board only has two possible addresses to work with, we constructed a process to send the sensor data over the serial port, mapping the active sensor to be read to the first address and the inactive sensors mapped to the second.

Every time a sensor needs to be read, we can switch it to the active address and send the required orientation data in the form of a quaternion (x, y, z, w) to the computer. This way we could bypass the two-sensor limit and have any number of them attached to the board as long as we have the space for it. Theoretically, we can have up to fourteen sensors per board, one for each digital pin.

**Fanning Out the Signal**  
In order to successfully connect multiple sensors to an Arduino board, each connection except for the A0 pin, which is used by each sensor to tell it what individual address it has been assigned, can be shared. Four total pins are fanned out to every MPU connected to the Arduino:

- 3.3V to each VCC - This connection provides power  
- GND to each GND - This connection is the ground which completes the circuit  
- Analog 4 to each SDA - The I2C bus data line for information transfer  
- Analog 5 to each SCL - The I2C bus clock line for synchronization

**From the Board to Unity**  
**Data Through the Serial Port**  
All data is sent to the computer through the serial port, a communication interface which information transfers in or out one bit at a time. Captured data from each sensor is assigned an id which is sent across the serial port along with its rotation values to be read by our script within Unity.

<table>
<thead>
<tr>
<th>Id</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.97862</td>
<td>-0.26417</td>
<td>0.27803</td>
<td>0.61859</td>
</tr>
<tr>
<td>1</td>
<td>0.97705</td>
<td>0.38397</td>
<td>1.22919</td>
<td>-0.38239</td>
</tr>
<tr>
<td>2</td>
<td>1.90729</td>
<td>1.35913</td>
<td>1.48285</td>
<td>-1.89764</td>
</tr>
<tr>
<td>3</td>
<td>-0.75726</td>
<td>1.87604</td>
<td>0.23627</td>
<td>-0.84680</td>
</tr>
<tr>
<td>4</td>
<td>0.48267</td>
<td>1.04260</td>
<td>0.68219</td>
<td>-0.76996</td>
</tr>
</tbody>
</table>

Raw quaternion values given directly by a sample of five sensors

**Calibration**  
In its raw form, the given quaternion values are difficult to work with because they are not oriented in the correct position needed to track on the mannequin. After each sensor is activated for the first time, calibration is needed.

To calibrate each rotation given by a sensor, the inverse of the quaternion generated is calculated. By multiplying this inverse by each sequential rotation, we can offset the model mannequin limb back to its default location to correctly track the sensor’s movement.

```csharp
// At startup:  
Quaternion q = ReadRotation(serialLine);  
offset[i] = Quaternion.Inverse(q);  
For each new quaternion q read:  
limbs[i].rotation = offset[i] * q;  
```

Where limbs is an array of limbs on the mannequin, q is the rotation read in and offset is an array of calibration offsets.

**Applications**  
We envision that this software could be used as an aid for skeletal animation or as an input device specifically for video game development.

Future variations this project could include easy adaptation for different models, wireless sensor connection, or upgrading the components to integrate more accelerometers into the system.

**References**  
Rowberg, Jeff. I2Cdevlib, 2018, GitHub, https://github.com/jrowberg/i2cdevlib

**Acknowledgements**  
This project was funded by the Blugold Fellowship program.