

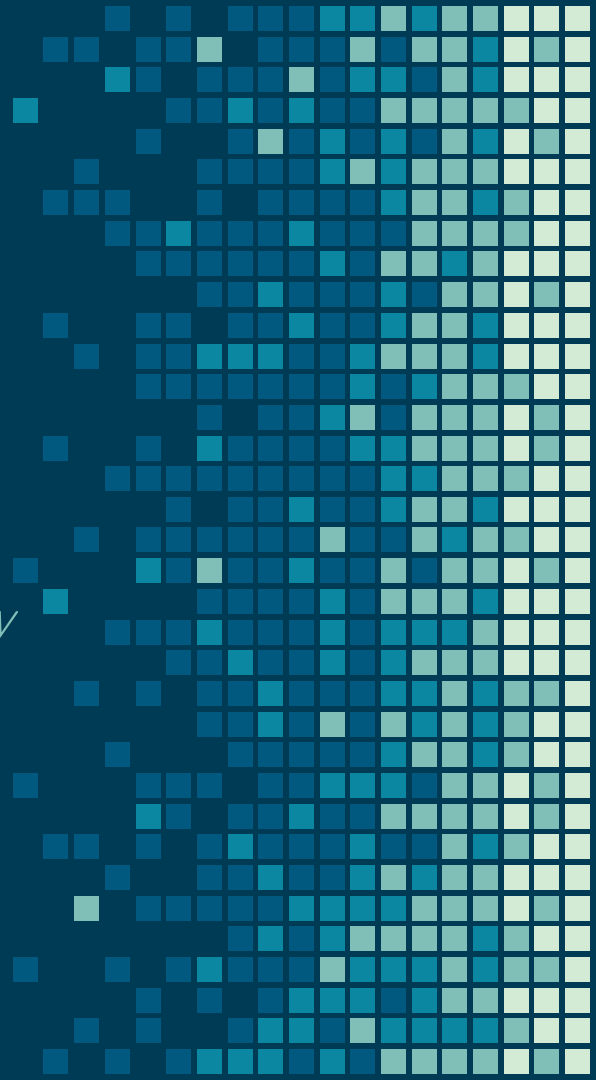
Virtual Reality vs Augmented Reality

With an implementation of navigating the maze in virtual reality

Mark Huffman

EE 444, Professor Hill

11/29/2017



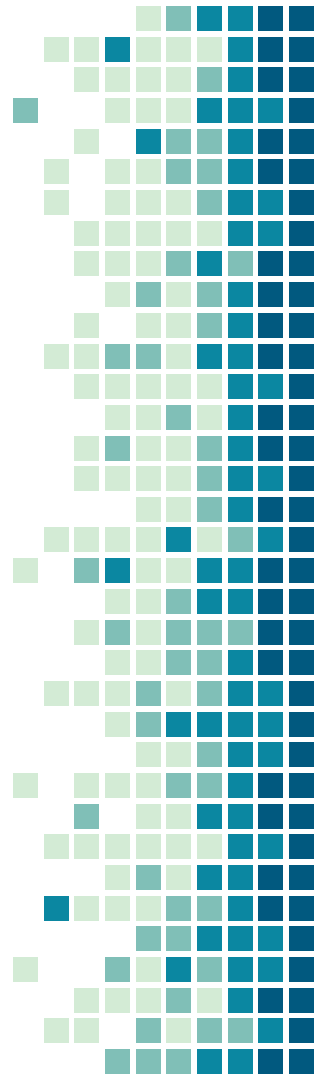
What are they?

Virtual Reality

A complete computer generated simulation, obscuring the user's vision, while allowing the user to interact in the simulation in a real or physical way

Augmented Reality

Computer generated images or simulation placed over the visible world, typically through a camera system or transparent displays



What are they?

Virtual Reality

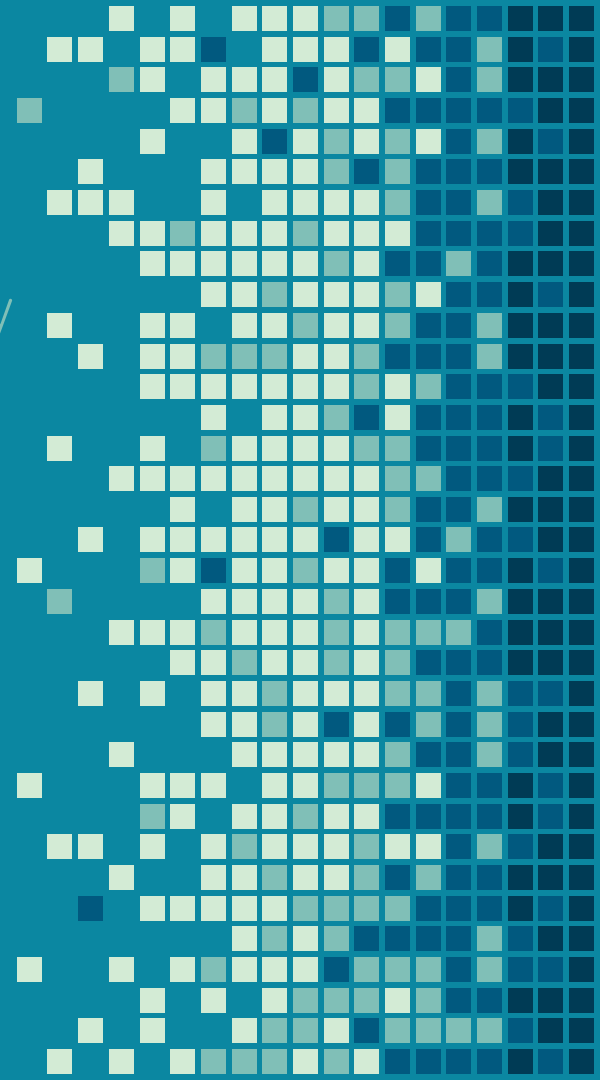


Augmented Reality



Augmented Reality Summary

- Projection Methods
- Recognition Methods



Projection Methods

- **External Camera Feed** - Using a connected camera, a computer generated image is generated over the the live view feed proved by an external camera
- **Transparent Display** - Solid Beam Splitter - used with a polarizing beam splitter to combine light from a LCOS microdisplay



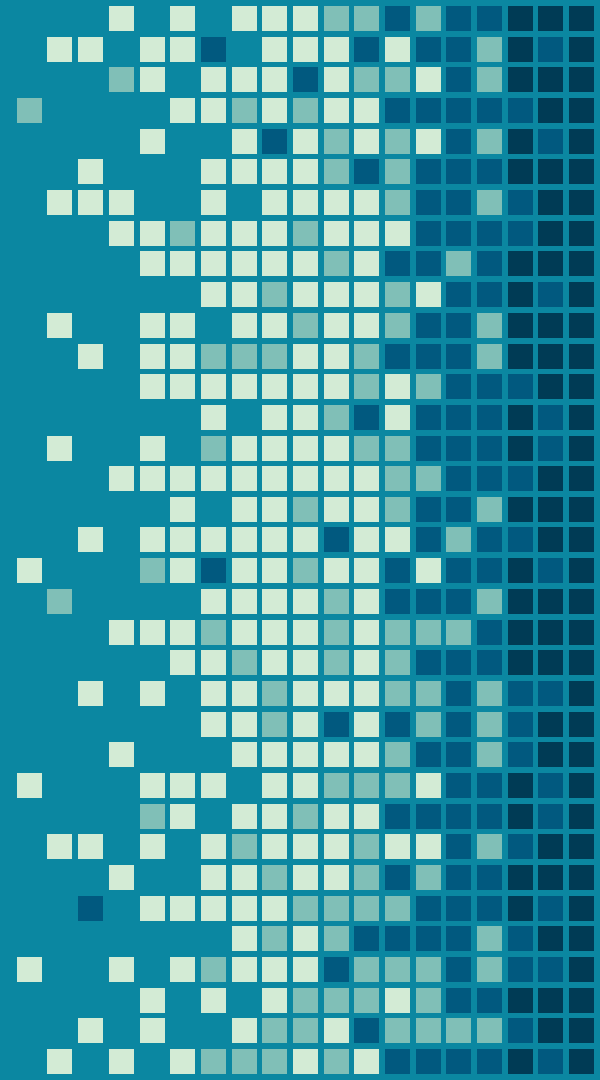
Recognition Methods

- **Location Based** - takes combined information from gps, digital compass and gyroscope to with a computer recognition method called SLAM (Simultaneous Localisation and Mapping)
- **Marker Based** -A simple visual marker like a QR code is identified and a known overlay is produced in its location, taking low compute power



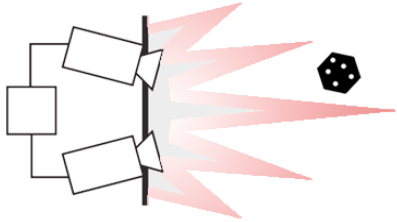
Virtual Reality Summary

- Tracking Methods
- Display Breakdown

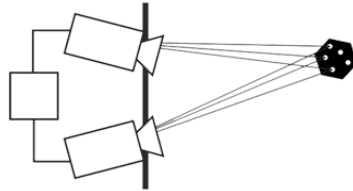


Tracking Methods

- **Optical/Consolation** - Reflective markers are placed on the object and camera emit IR light reflections are captured by the camera with an IR pass filter
- **IR Tracking/Lighthouse** - IR beacon(s) sweep the area vertically or horizontally in IR light, the tracking device captures the timing



The object is lit using near IR light

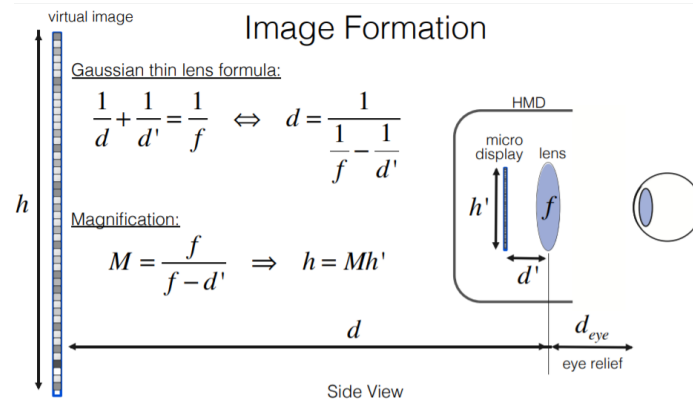
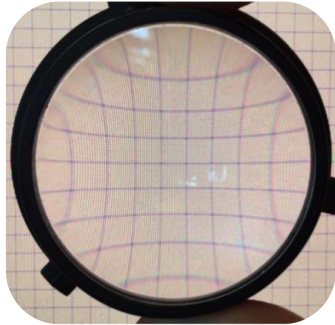


Retro-reflective markers reflect back



Optics and Displays

- All HMDs are composed of a set of lens and micro display panel(s)
- The lens magnify the screen giving the illusion of a larger virtual image
- Lens create distortion, in the (x,y) plane that is corrected in software before being displayed



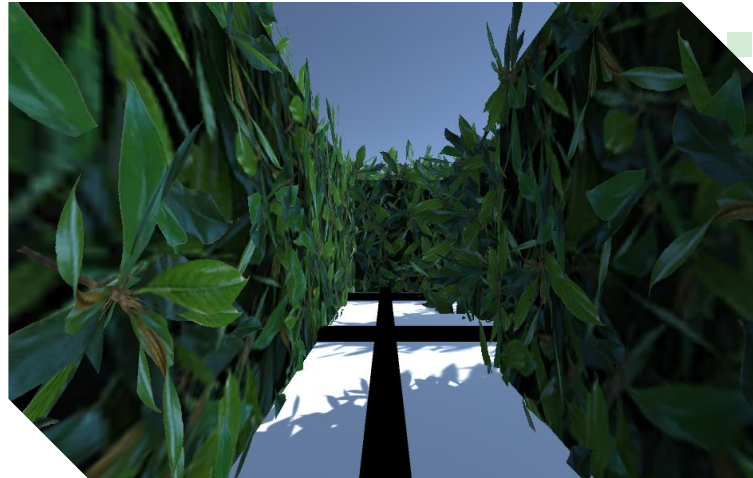
Practical Experiment

- Traveling through a virtual maze in real time
- Version 1.0 (Proof of concept)
- Version 2.0 (Class implementation)



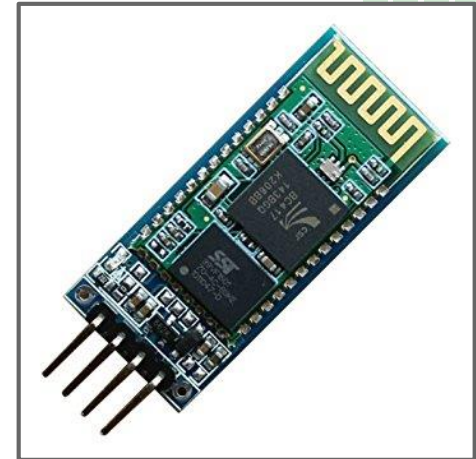
3D Hedge maze – Version 1.0

- Using Unity and public assets a small 3D version of the maze was created on a simple grid pattern



HC-06 Communication to PC

- Adding a HC-06 Bluetooth Module to the bot allowed coordinates to be sent back to the PC in realtime based on the current location within the physical maze
- This would then control the movement of the camera within unity



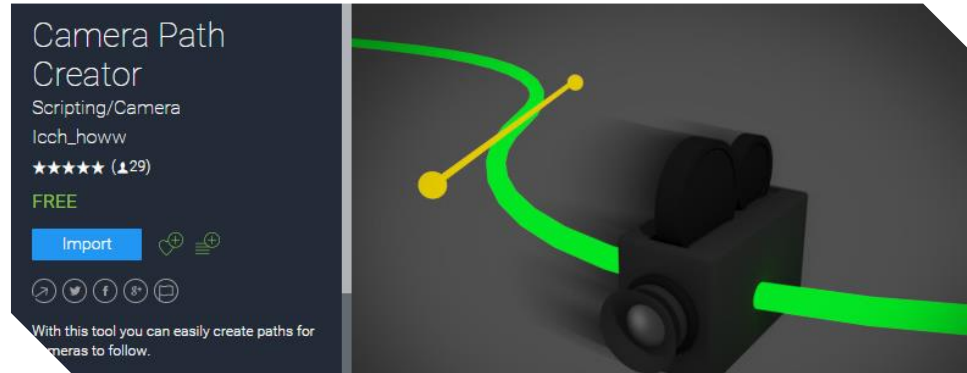
General Methodology - Version 1.0

1. Connect to HC-06 (Using known COM port)
2. Start the bot, and trigger the first camera movement
3. When the camera arrives at first target ping the bot a wait for a response to continue
4. Repeat for each additional camera check point
1. Arduino: Respond to all requests as "busy" or "advance" camera depending on turning action



Camera Movement Handled by CPC

- A free unity camera asset
- Modified to be triggered by receiving commands from arduino
- Has built in functions to handle complex camera rotation and movement through 3D space

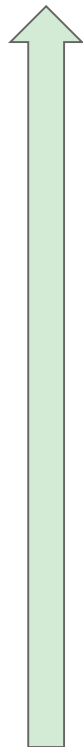


Unity code to trigger camera Movement

```
void blueUpdate()
{
    // Make sure bluetooth is connected
    if (!spOut.IsOpen)
    {
        Debug.Log("Open the port!");
        spOut.Open();
    }

    int trys = 0;
    Boolean communication = false;
    String message = "";

    // Loop to allow fails in communication
    while (!communication)
    {
        try
        {
            // Send "Ready" command to bot
            spOut.Write("R");
            System.Threading.Thread.Sleep(100);
            // Read incoming message from bot
            message = spOut.ReadLine();
            communication = true;
        }
        catch (IOException)
        {
            // Failure on reading message
            // After 10 failed attempts giveup
            if (trys > 10)
                communication = true;
            // wait and then try again
            System.Threading.Thread.Sleep(100);
            trys++;
        }
    }
}
```



```
}
    Debug.Log("Received: "+message);

    // If message is "-" continue moving camera
    if (message.EndsWith("-"))
        WaitFor = false;
    else
        System.Threading.Thread.Sleep(200); // Else wait a bit longer, bot not ready
}

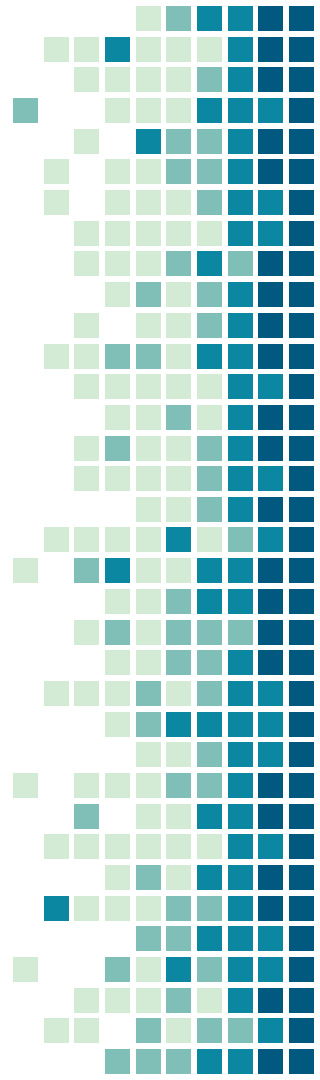
public void connectBluetooth()
{
    // Setup serial bluetooth communication
    spOut = new SerialPort("COM7", 9600, Parity.None, 8, StopBits.One);
    spOut.Handshake = Handshake.None;
    spOut.ReadTimeout = 3000; // Note* Must have a long time out
    spOut.WriteTimeout = 500;

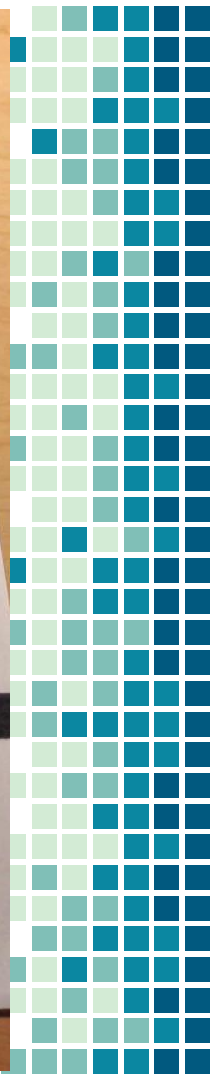
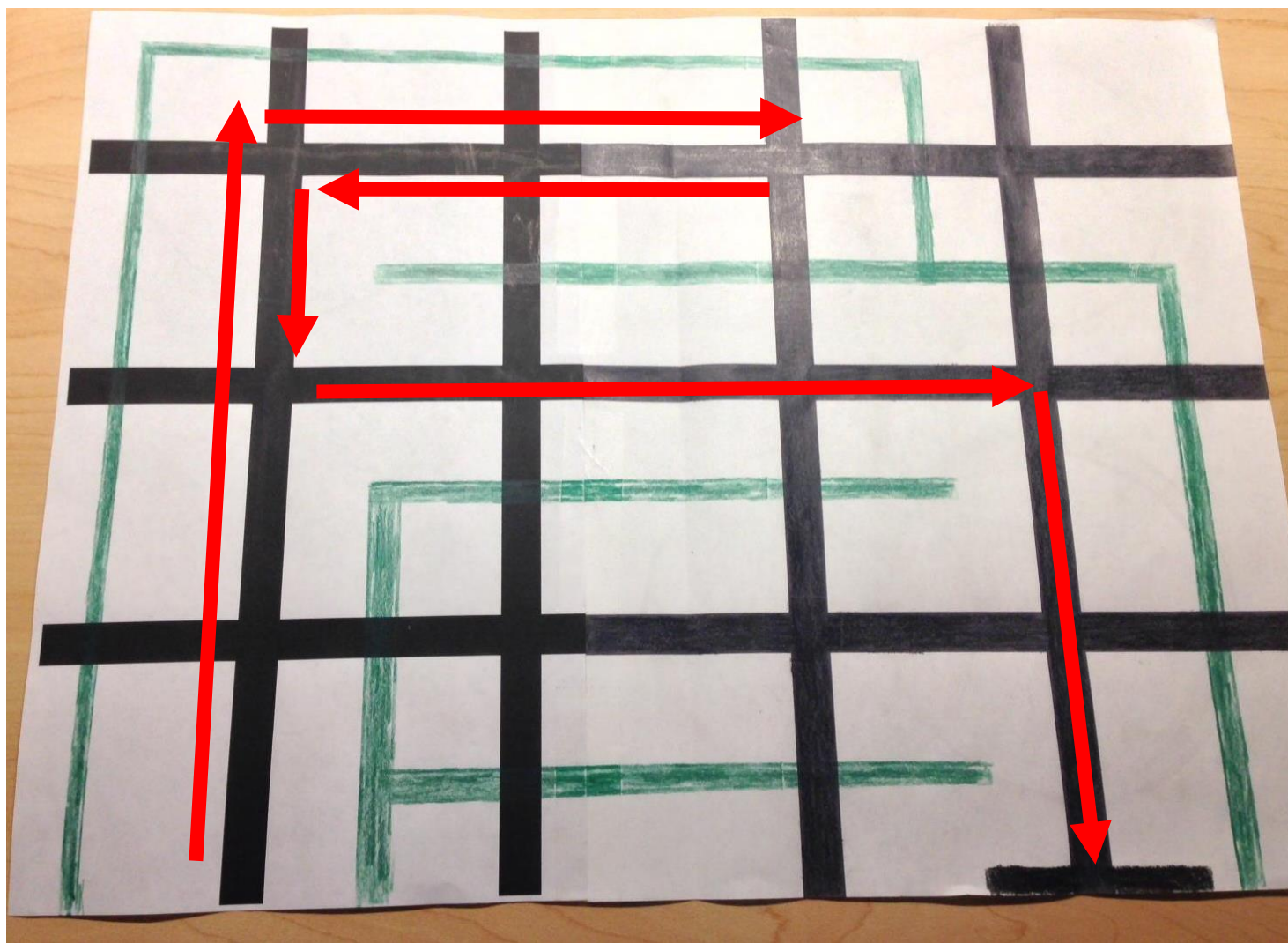
    // Open the port for communication
    if (!spOut.IsOpen)
    {
        Debug.Log("Open the port!");
        spOut.Open();
    }
    Debug.Log("Connected!");
}
```



Arduino Trigger Camera Movement

```
void blueToothCheck(){  
  // Check if a signal was received  
  if (blueTooth.available()) {  
    // Read the incoming signal  
    char c = blueTooth.read();  
    //Serial.print("BT read: "); Serial.println(c);  
    // If 'R' received then Unity is expecting the next command  
    if(nextStep && c == 'R'){  
      // Send Next "Move" Command  
      blueTooth.println("-"); // Move clip further  
      // clear the flag  
      nextStep = false;  
      // Initial Startup received  
      if(start){  
        motorControl("Go");  
        start = false;  
      }  
    }  
  }  
  // Received Signal but not read for next action  
  // Tell Unity to wait  
  blueTooth.println("w"); // Wait bot not ready  
}  
}
```

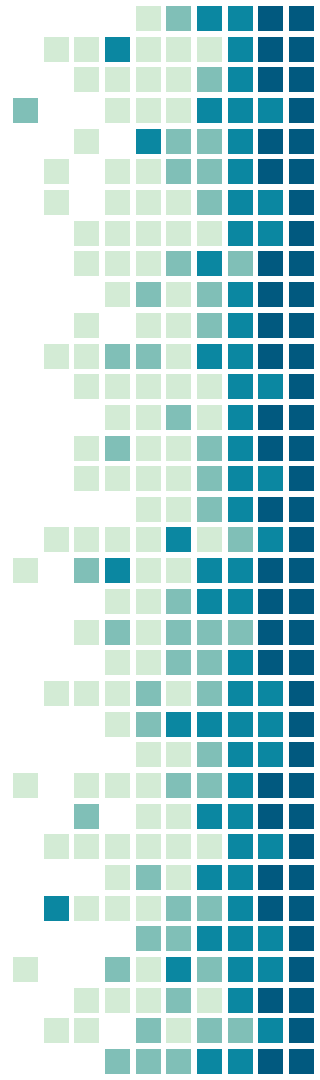




Click to view video

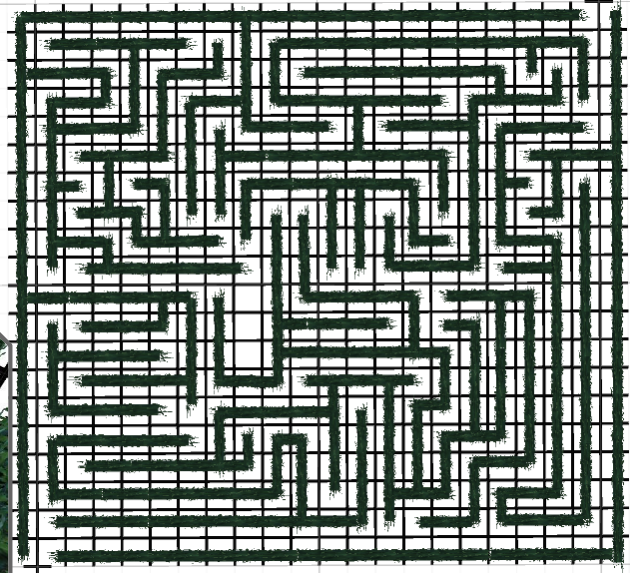
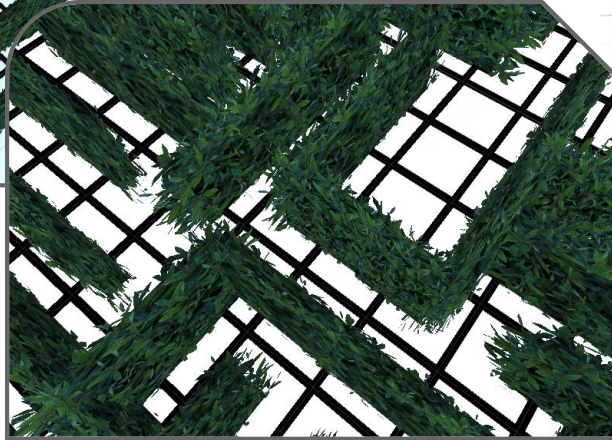
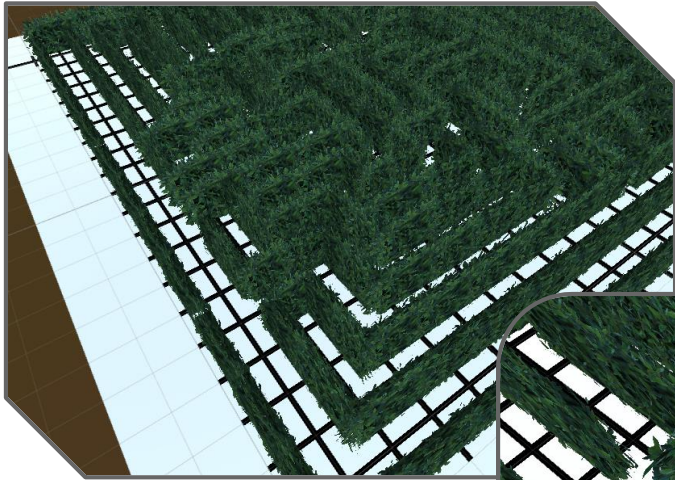
Next Step

- This proof of concept shows that this method could be adapted to the actual maze and then be redesigned in such a way that any route could be taken through the maze
 - Feedback turn, direction and coordinates to Unity and then move the camera as defined by these received actions
- Design the code to be adapted to any bot and take any path through the maze



3D Hedge Maze Version 2.0

- 3D Version of our full Hedge Maze



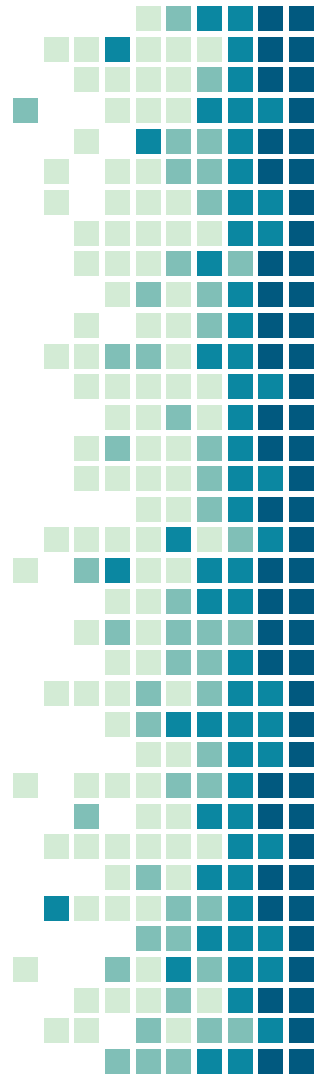
General Methodology – Version 2.0

1. Connect to HC-06 (By scanning all COM ports)
2. Start the bot and receive timing data
3. Receive a camera command from arduino
 - a. Indicating a Left, Right or U-turn

Or a distance to travel forward in “squares”

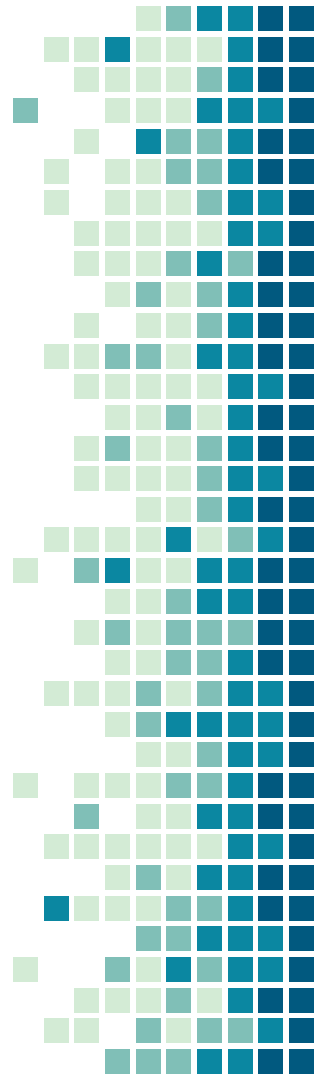
Move the camera as appropriate to the received commands and keep track of location and orientation

Ask for new direction instructions when action is completed



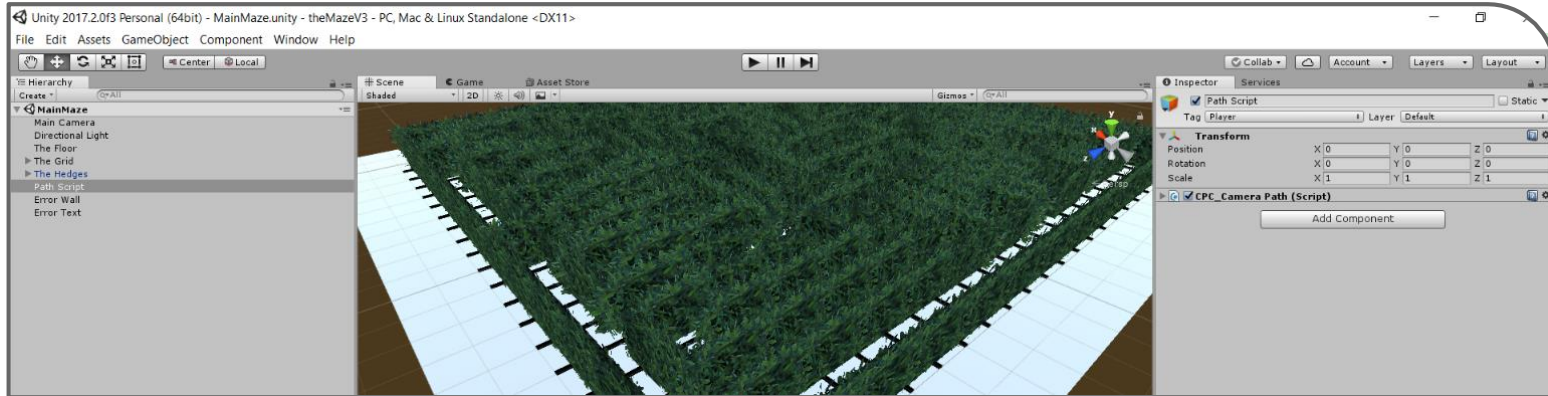
Unity Code – Breakdown

- Heavily Modified version of CPC Script
- void connectBluetooth()
 - Scan all com ports and find one with response
- void initializeArduinio()
 - Tell bot to start moving and interpret timing information
- void blueUpdate()
 - Ping bot then receive next action commands and decode
- float timePerSegmentDef(int type, int num)
 - Determines time of each action based on timing values
- void updatePos(int squares, int turn)
 - Updates in maze position based on new request and current position
- IEnumerator FollowPath(float time)
 - Calls all action functions and is the calling “process”



Unity Code – Breakdown Continued

- Vector3 GetBezierPosition(CPC_Point current, CPC_Point next, float time)
 - Moves the camera vector based on position change and time
- private Quaternion GetLerpRotation(CPC_Point current, CPC_Point next, float time)
 - Rotates the camera vector based on quaternion math

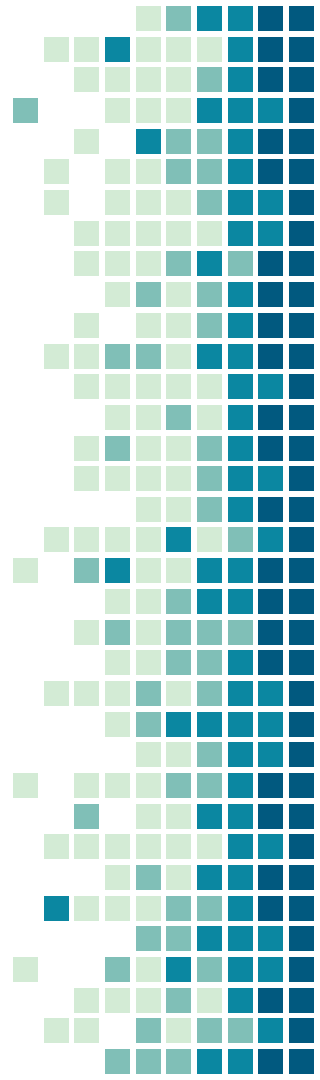


Unity Code - Communication

```
public void connectBluetooth()
{
    // Setup serial bluetooth communication
    int i = 1;
    Boolean found = false;

    // Scan all COM ports looking for particular response
    while (!found && i < 50)
    {
        try
        {
            spOut = new SerialPort(@"\\.\COM" + i, 9600, Parity.None, 8, StopBits.One); //Must match communication standard
            spOut.Handshake = Handshake.None;
            spOut.ReadTimeout = 3000; // Note* Must have a long time out
            spOut.WriteTimeout = 500;

            // Open the port for communication
            if (!spOut.IsOpen)
            {
                Debug.Log("Open the port!");
                spOut.Open();
            }
            spOut.Write("I"); //write expected output
            System.Threading.Thread.Sleep(100);
            // Read incoming message from bot
            String message = spOut.ReadLine();
            found = true;
        }
        catch (Exception)
        {
            i++;
        }
    }
    Debug.Log("Connected to COM"+i+"!");
    if(i < 50)
        selectedCamera.transform.position = Startpoint.position;
}
```



Unity Code - Communication

```
void initializeArduino()
{
    // Make sure bluetooth is connected
    if (!spOut.IsOpen)
    {
        Debug.Log("Open the port!");
        spOut.Open();
    }

    int trys = 0;
    Boolean communication = false;
    String message = "";

    // Loop to allow fails in communication
    while (!communication)
    {
        try
        {
            // Send "Ready" comand to bot
            spOut.Write("C");
            System.Threading.Thread.Sleep(10);
            // Read incoming message from bot
            message = spOut.ReadLine();
            communication = true;
        }
        catch (IOException)
        {
            // Failure on reading message
            // After 10 failed attempts giveup
            //Debug.Log("ERROR");
            if (trys > 10)
            {
                communication = true;
                Debug.Log("ERROR Communication Failed!");
            }
            // wait and then try again
            System.Threading.Thread.Sleep(100);
            trys++;
        }
    }
}
```




```
if (message == null)
    return;
Debug.Log("Recived: " + message);
// Break out timings
timeValue[0] = float.Parse(message.Substring(0, 1) + "." + message.Substring(1, 1)); // message[0].[message[1]f
timeValue[1] = float.Parse(message.Substring(2, 1) + "." + message.Substring(3, 1));
timeValue[2] = float.Parse(message.Substring(4, 1) + "." + message.Substring(5, 1));
}
```

Unity Code - Communication

```
void blueUpdate()
{
    // Make sure bluetooth is connected
    if (!spOut.IsOpen)
    {
        Debug.Log("Open the port!");
        spOut.Open();
    }

    int tries = 0;
    Boolean communication = false;
    String message = "";

    // Loop to allow fails in communication
    while (!communication)
    {
        try
        {
            // Send "Ready" comand to bot
            spOut.Write("R");
            System.Threading.Thread.Sleep(10);
            // Read incomming message from bot
            message = spOut.ReadLine();
            communication = true;
        }
        catch (IOException)
        {
            // Failure on reading message
            // After 10 failed attempts giveup
            //Debug.Log("ERROR");
            if (tries > 10)
            {
                communication = true;
                Debug.Log("ERROR Communication Failed!");
            }
            // wait and then try again
            System.Threading.Thread.Sleep(10);
            tries++;
        }
    }
    if (message == null)
        return;
    Debug.Log("Recived: "+message);
}
```



```
// Check current camera Angle (For intial)
Currentpoint.rotation = Quaternion.Euler(0, currentAngle, 0);

if (message.StartsWith("S"))
{
    // Bot is going straight
    // Get Number of blocks from message
    Currentpoint = Nextpoint;
    int squares = 0;
    if(message.Length == 2)
        squares = int.Parse(message.Substring(1, 1));
    else if(message.Length == 3)
        squares = int.Parse(message.Substring(1, 2));
    updatePos(squares, 0);
    Nextpoint = new CPC_Point(new Vector3(xCord, 6, zCord), new Quaternion());
    Nextpoint.rotation = Quaternion.Euler(0, currentAngle, 0);
    nextInstuction = 0;
    strightCount = squares;
    WaitFor = false;
}
```

Unity Code - Communication

```
else if (message.StartsWith("T"))
{
    // Bot is going to turn
    // Get turn type from message
    Currentpoint = Nextpoint;
    if (message.EndsWith("A")) {
        nextInstuction = 2; // Turning around
        updatePos(0, 3);
        Nextpoint = new CPC_Point(new Vector3(xCord, 6, zCord), new Quaternion());
        currentAngle -= 180;
        Debug.Log("New Angle: " + currentAngle);
        Nextpoint.rotation = Quaternion.Euler(0, currentAngle, 0);
    } else if (message.EndsWith("L")) {
        nextInstuction = 1; // Turning Left
        updatePos(0, 2);
        Nextpoint = new CPC_Point(new Vector3(xCord, 6, zCord), new Quaternion());
        currentAngle -= 90;
        Debug.Log("New Angle: " + currentAngle);
        Nextpoint.rotation = Quaternion.Euler(0, currentAngle, 0);
    }
    else if (message.EndsWith("R")) {
        nextInstuction = 1; // Turning Right
        updatePos(0, 1);
        Nextpoint = new CPC_Point(new Vector3(xCord, 6, zCord), new Quaternion());
        currentAngle += 90;
        Debug.Log("New Angle: " + currentAngle);
        Nextpoint.rotation = Quaternion.Euler(0, currentAngle, 0);
    }
    WaitFor = false;
}
```

```
else if (message.StartsWith("W"))
{
    // Bot not read to send next direction
    System.Threading.Thread.Sleep(100); // Else wait a bit longer, bot not ready
}
else if (message.StartsWith("-"))
{
    // Test Message
    //WaitFor = false;
}
```



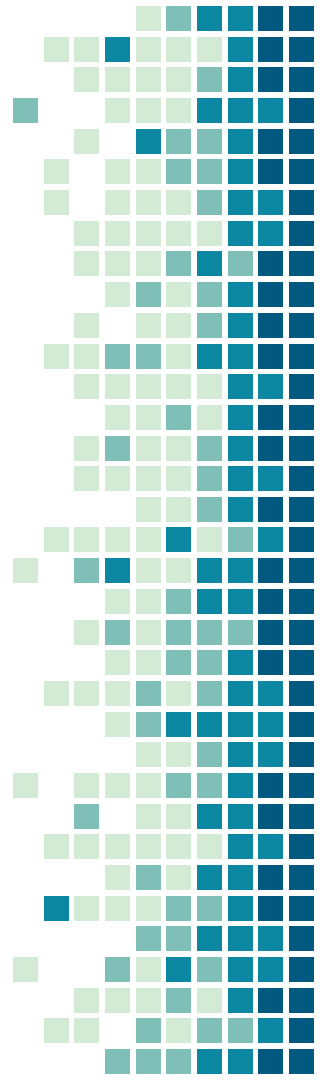
Unity Code - Camera Control

```
IEnumerator FollowPath(float time)
{
    initializeArduino();
    while (!endOfpath)
    {
        currentTimeInWaypoint = 0;
        if (WaitFor == false)
        {
            WaitFor = true;
            while (currentTimeInWaypoint < 1)
            {
                if (!paused)
                {
                    if (!startup)
                    {
                        spOut.Write("C");
                        startup = true;
                    }
                    // Determine Time of Segment
                    currentTimeInWaypoint += Time.deltaTime / timePerSegmentDef(nextInstruction, strightCount);
                    // Move Camera to desired coordinates
                    selectedCamera.transform.position = GetBezierPosition(Currentpoint, Nextpoint, currentTimeInWaypoint);
                    selectedCamera.transform.rotation = GetLerpRotation(Currentpoint, Nextpoint, currentTimeInWaypoint);
                }
                yield return 0;
            }
        }
        else
        {
            yield return new WaitForSeconds(0.2f);
            blueUpdate();
        }
    }
    StopPath();
}
```

Arduino Code – Implementation

Requirements

- Your bot must stop before making a turn and after completing a turn
 - Communication commands will hold up the rest of your code and your bot will wonder off the maze otherwise.
- Completion of Lab 5-6 (Whichway and navigation out off the maze instructions)
- HC-06 or similar setup with bluetooth to arduino, serial communication



Arduino Code – Implementation Instructions

- Add "unity_commands.ino" to your project
- Modify countTillNextTurn() to save whatever variables are used in your which way to determine your path (AKA hold1,hold2)
- Add to your main file:
 - #include <SoftwareSerial.h>
 - SoftwareSerial blueTooth(txPin, rxPin); // define to connected pins
- At the very end of setup() add:
 - connectUnity();
 - int times[] = {2,0,2,4,4,0}; // Modify time values
 - setupUnity(times);

Note*

0,1 [#.##f] (Time for a single straight block)

2,3 [#.##f] (Time for a 90 degree turn)

4,5 [#.##f] (Time for a 180 degree turn)

Example: {1, 2 , 0, 0, 0, 0}

1.2f = 1.2 seconds for straight block

Arduino Code – Implementation Instructions

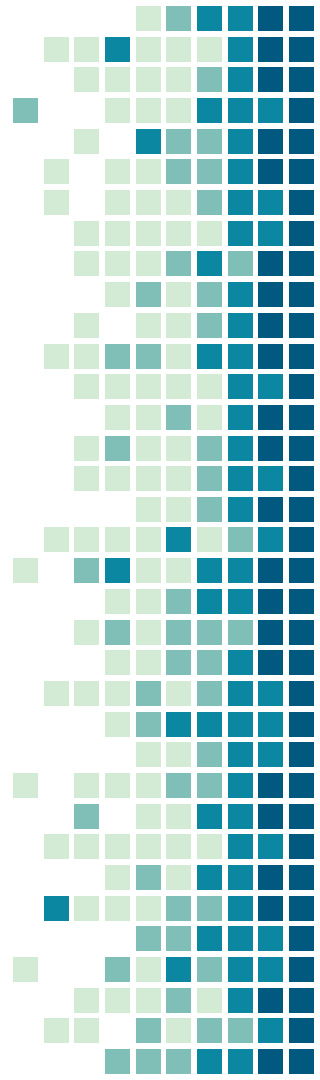
- In the main loop() add the following before anything
 - *if(start){*
 - *// Start Motors*
 - *unityUpdate(countTillNextTurn(mazeBot));*
 - *// ADD: Call whatever command starts Motors here*
 - *start = false;*
 - *}*
 - *}else{ // Everything else in loop()*
- At the very end of loop() add: *checkUnity();*
- Call *unityUpdate(countTillNextTurn(mazeBot));* **After every Turn**
- Call *unityUpdate(getNextTurnAction(mazeBot));* **Before every Turn**

Note*

Must be called after motors have stopped!
Will hold up ALL code until a response from unity is received.

Arduino Code – unity_commands.ccp

- void checkUnity()
 - Called by the main program loop, responds to unity if a command is ever received early as “Wait!”
- void connectUnity()
 - Called in setup, holds until an initialize ping is received from unity (Correct COM port selected)
- void setupUnity(in timings[])
 - Called next in setup, holds until unity is done loading then passes timing
- void unityUpdate(String command)
 - Called before the start of a turn and just after the completion of a turn, holds till unity pings “ready” (Called with the next two functions)
- String countTillNextTurn(MyRobot mazeBot)
 - Placed in unityUpdate() after completing a turn, calculates number of squares to next to turn action using virtual instructions and whichway
- String getNextTurnAction(MyRobot mazeBot)
 - Placed in unityUpdate() before starting a turn, sends command based on current virtual .turn instruction



Arduino Code – Communication

```
void checkUnity(){
  // Responds to all Unity requests as "Bot" not ready
  // Keeps connection live
  if (blueTooth.available()) {
    // Read the incoming signal
    char c = blueTooth.read();
    if (c == 'R') {
      // Not Ready to send next instruction
      blueTooth.println("W"); // tell unity to wait
    }
  }
}

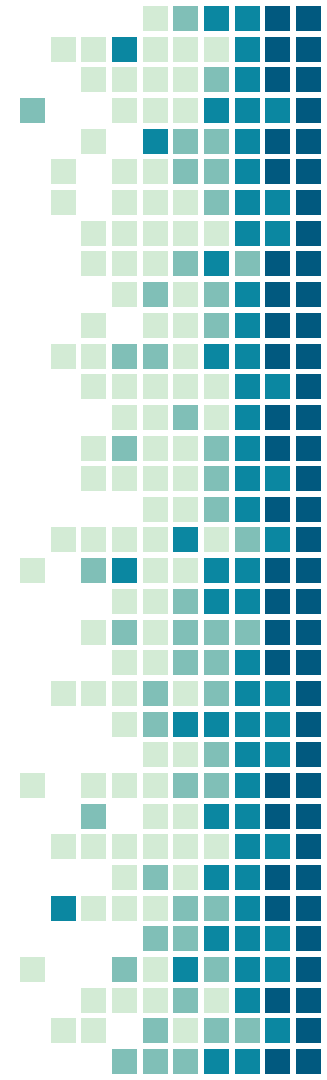
void connectUnity(){
  // Holds here till unity replies with a connection signal
  // Note* Bot must not be moving when calling this command!
  boolean sentcommand = false;
  while(!sentcommand){
    if (blueTooth.available()) {
      // Read the incoming signal
      char c = blueTooth.read();
      if (c == 'I'){
        // Send connection valid signal!
        blueTooth.println("-");
        sentcommand = true;
      }
    } else {
      delay(10);
    }
  }
}
```

```
void setupUnity(int timings[]){
  // Holds here till unity replies with a ready signal
  // Note* Bot must not be moving when calling this command!
  // --- Unity Timmings ---
  // - 0,1 [#.##] (Time for a single straight block)
  // - 2,3 [#.##] (Time for a 90 degree turn)
  // - 4,5 [#.##] (Time for a 180 degree turn)
  boolean sentcommand = false;
  while(!sentcommand){
    if (blueTooth.available()) {
      // Read the incoming signal
      char c = blueTooth.read();
      if (c == 'C') {
        // Send Timming Initialize Command
        String command = "";
        command = command + timings[0] + timings[1] + timings[2] + timings[3] + timings[4] + timings[5];
        blueTooth.println(command);
        sentcommand = true;
      }
    } else {
      delay(10);
    }
  }
}
```



Arduino Code – Communication

```
void unityUpdate(String command){  
  // Waits for unity to send request command, then responds with next action  
  // Note* Bot must not be moving when calling this command!  
  // --- Unity Commands: ----  
  // - S# (Continue straight for # number of intersections)  
  // - TR (Turn right 90 degrees)  
  // - TL (Turn Left 90 degrees)  
  // - TA (Turn Around 180 degrees)  
  boolean sentcommand = false;  
  while(!sentcommand){  
    if (blueTooth.available()) {  
      // Read the incoming signal  
      char c = blueTooth.read();  
      if (c == 'R') {  
        // Send Next "Move" Command  
        blueTooth.println(command);  
        sentcommand = true;  
      }  
    }else{  
      delay(10);  
    }  
  }  
}
```



Arduino Code – Next Action

```
String countTillNextTurn(MyRobot mazeBot){
  // Gets squares count to next intersection based on maze coordinates and whichway command
  // Make new instance of MazeBot
  // Start at same position
  // Simulate continued movement till turn required.
  MyRobot tempBot{mazeBot.dir, mazeBot.turn, mazeBot.row, mazeBot.col, mazeBot.room, 0x00};
  uint8_t squares = 0;
  // Save Whichway Counters
  uint8_t hold1 = physicalBot.routeN;
  uint8_t hold2 = physicalBot.routeL;

  while(tempBot.turn == 0x00 || squares == 0){

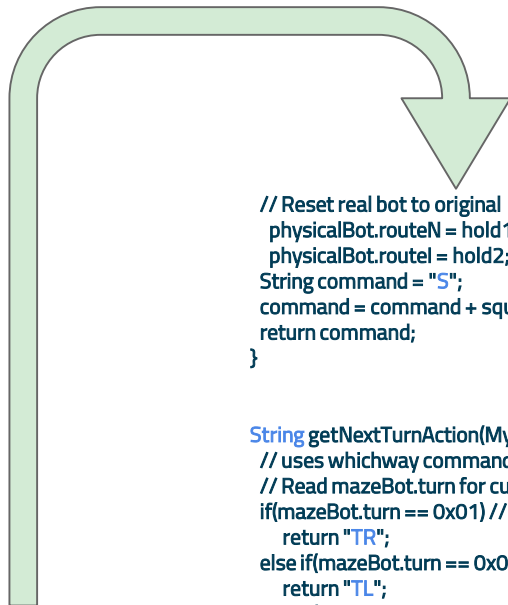
    /* Update dir by turing in the maze */
    tempBot.dir = turnInMaze(tempBot);

    /* Reference return variable */
    uint8_t *updateValues;

    /* Update col and row loaction */
    updateValues = stepInMaze(tempBot);
    tempBot.row += *(updateValues);
    tempBot.col += *(updateValues + 1);

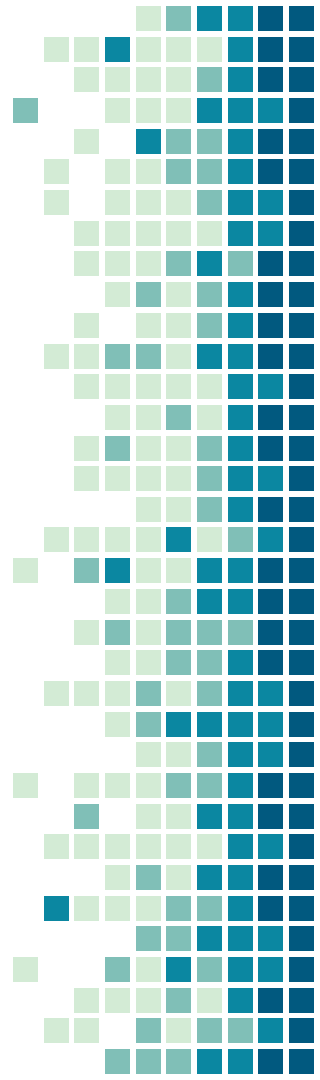
    /* Update room and bees */
    updateValues = roomInMaze(tempBot);
    tempBot.room = *(updateValues);
    tempBot.bees = *(updateValues + 1);

    /* Determine if a turn is needed */
    tempBot.turn = whichWay(tempBot);
    squares++;
  }
}
```



```
// Reset real bot to original
physicalBot.routeN = hold1;
physicalBot.routeL = hold2;
String command = "S";
command = command + squares;
return command;
}
```

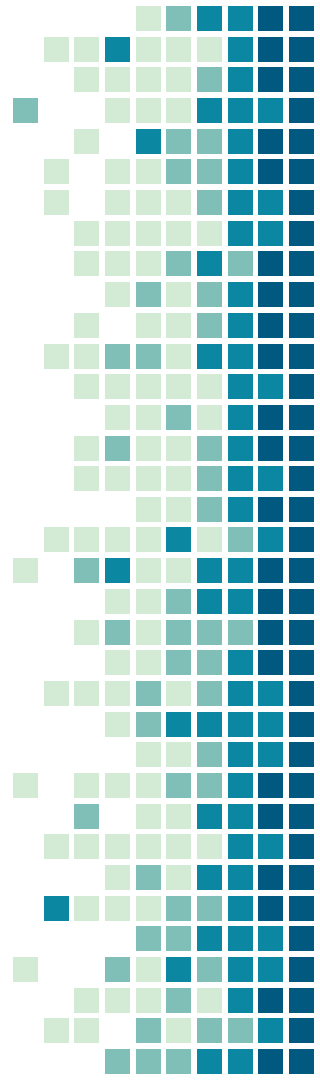
```
String getNextTurnAction(MyRobot mazeBot){
  // uses whichway command and determines turn direction
  // Read mazeBot.turn for current turn instruction
  if(mazeBot.turn == 0x01) // Turn right
    return "TR";
  else if(mazeBot.turn == 0x02) // Turn left
    return "TL";
  else if(mazeBot.turn == 0x03) // Turn Around
    return "TA";
  return "SO"; // error don't move anywhere?
}
```





Possible Future Updates

- This could easily be improved to work for the 3Dot however the serial communication portion would need to be rewritten into USART
- Visual improvements could be made to the maze
- Currently any changes to the maze have to be manually built, an automated mode would be a neat addition
- A path indicator/path taken could be a cool visual addition



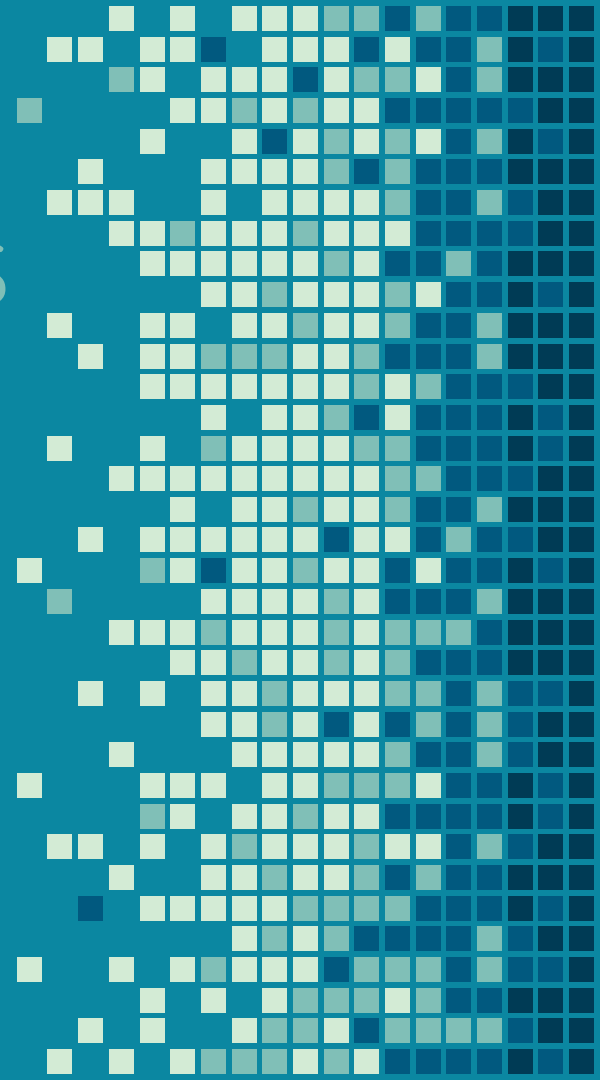
Issues Experienced

- Commands are not always received!
 - ReadTimeout error very likely
 - Could be serial.print
- HC-06 setup is very important and instructions can be inconsistent depending on source
 - Pin configuration arduino bluetooth setup
- Camera manipulation in Unity can be complex



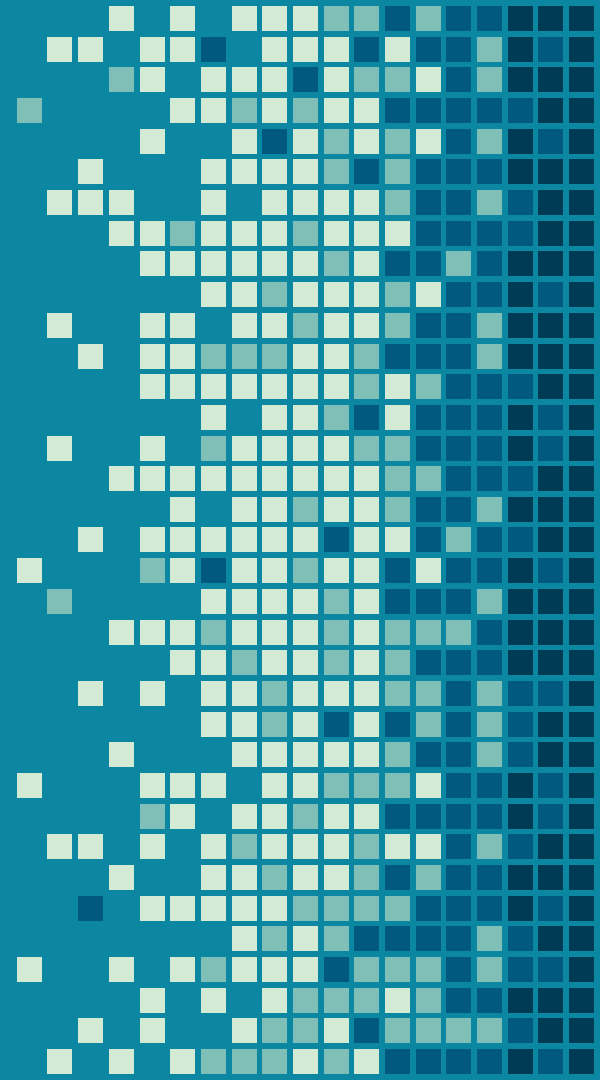
Appendix – Further Details

- Augmented Reality
- Virtual Reality



Augmented Reality

- Projection Methods
- Recognition Methods
- Practical use Cases



Projection Methods – External Camera Feed

- Using a connected camera, a computer generated image is generated over the the live view feed proved by an external camera
 - Marker Based, Superimposition
- Currently a native feature in Windows 10 (Mixed Reality Viewer)



-



Projection Methods – Transparent Display

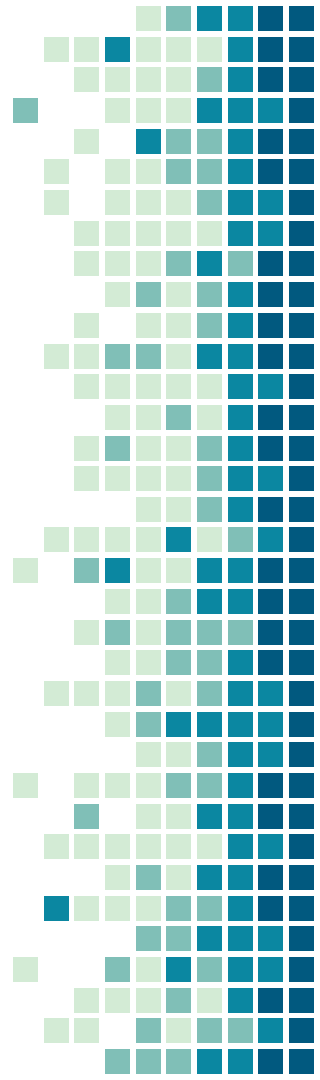
- Semi-spherical combiner – Act as a magnifying mirror moving the focus point



- Titled Thin Flat – Tilted combiner that's typically very large and cheaper to produce

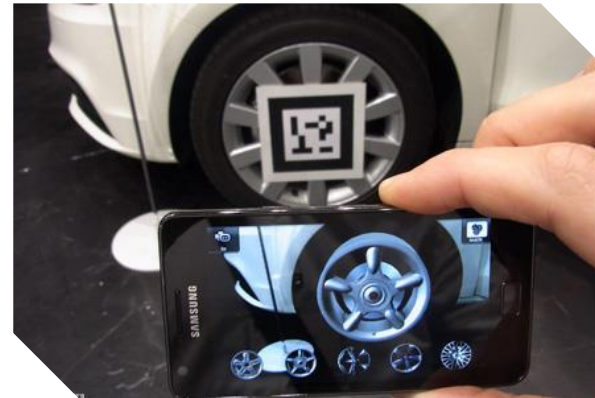
Recognition Methods – Location Based

- One of the most used methods, takes combined information from gps, digital compass and gyroscope to display relevant information
- This is then combined with a computer recognition method called SLAM (Simultaneous Localisation and Mapping)
 - This method runs complicated depth calculations and creates a digital map of the environment and then compares it to known mapped environments



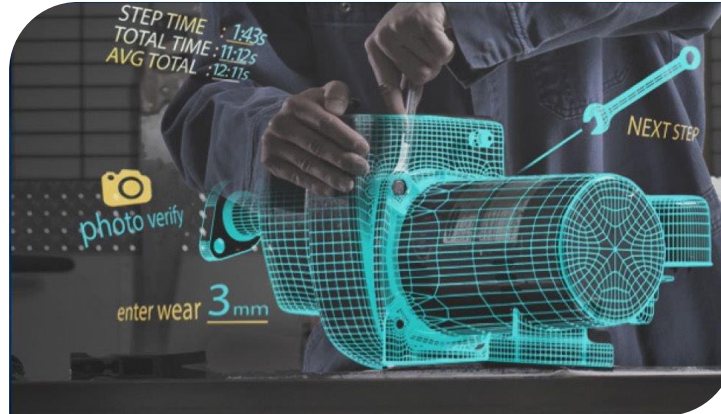
Recognition Methods – Marker Based

- A simple visual marker like a QR code is identified by a device's camera and a known overlay is produced in its location instead
- Takes far less computation than markless
- The markers are typically easily distinguishable for any camera system



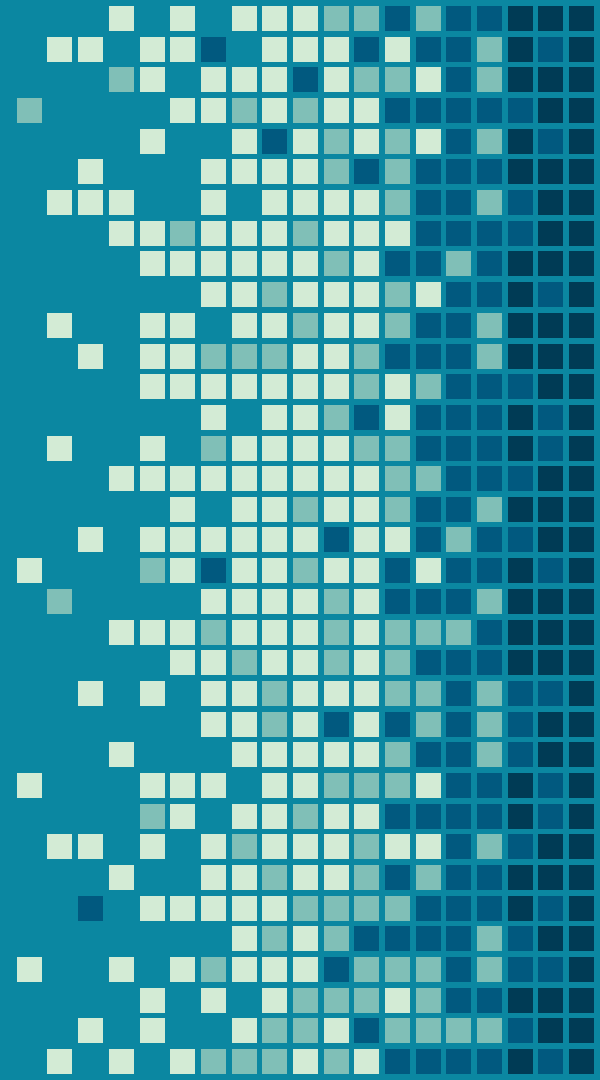
Practical Use Cases

- CAD design and collaboration
- Repair and maintenance instructions
- Military training
- Surgery guidance and access to references
- Collision guidance
- GPS navigation
- Job Training



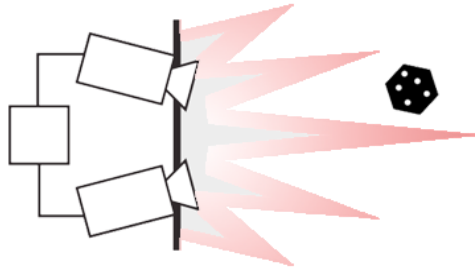
Virtual Reality

- Tracking Methods
- Display Breakdown
- Practical use Cases

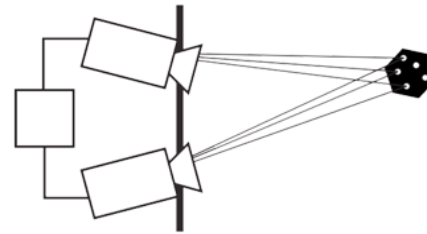


Tracking Methods - Optical/Constellation

- Reflective markers are placed on the object and camera emit IR light and the reflections are captured by the camera with an IR pass filter
- Multiple markers and cameras can be used to track an object's orientation and location within 3D space



The object is lit using near IR light



Retro-reflective markers reflect back

Tracking Methods - IR Tracking/Lighthouse

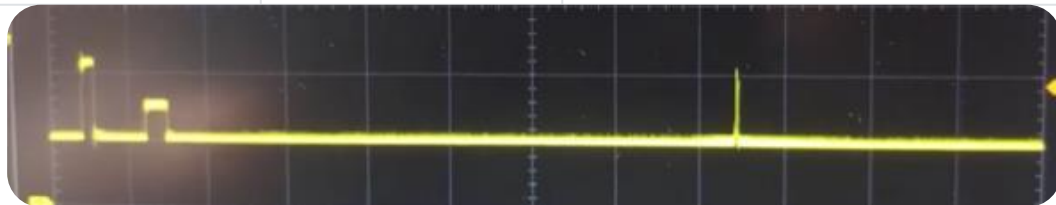
- A IR beacon strobes the space at 60 times per second then one of two spinning IR beacons sweep the area vertically or horizontally alternating
- The sensors located on the tracked object can then calculate their known position in 3D space using the fixed reference of the beacon



IR Tracking/Lighthouse - Breakdown

- 3D position can be updated every 4 cycles
cycles are 8.333ms exactly 120Hz

| Pulse start, μs | Pulse length, μs | Source station | Meaning |
|----------------------------|-----------------------------|----------------|--|
| 0 | 65–135 | A | Sync pulse (LED array, omnidirectional) |
| 400 | 65-135 | B | Sync pulse (LED array, omnidirectional) |
| 1222–6777 | ~10 | A or B | Laser plane sweep pulse (center=4000 μs) |
| 8333 | | | End of cycle |

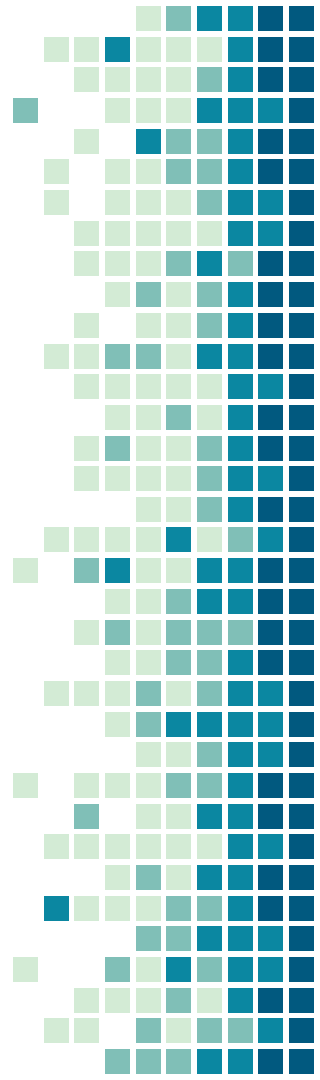


IR Tracking/Lighthouse - Calculations

- 3D position then determined by solving:
 - Known Base Station orientation and location $(x, y, z, \phi, \theta, \psi)$

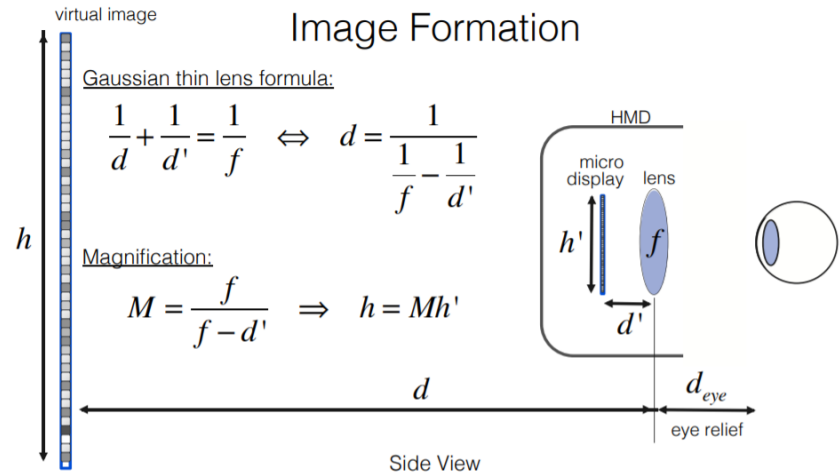
$$\hat{v}_{nk} = M_k \begin{bmatrix} 0 \\ \cos(\theta_{nk}) \\ \sin(\theta_{nk}) \end{bmatrix} \times \begin{bmatrix} \cos(\phi_{nk}) \\ 0 \\ -\sin(\phi_{nk}) \end{bmatrix}$$

- Finding the point of intersection of the two lines from current readings allows the vector position to be found from the reference locations
- Allows millimeter precision within a 3D space



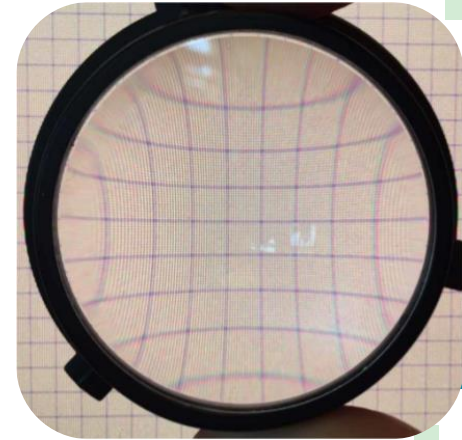
Optics and Displays

- All HMDs are composed of a set of lens and micro display panel(s)
- The lens magnify the screen giving the illusion of a larger virtual image



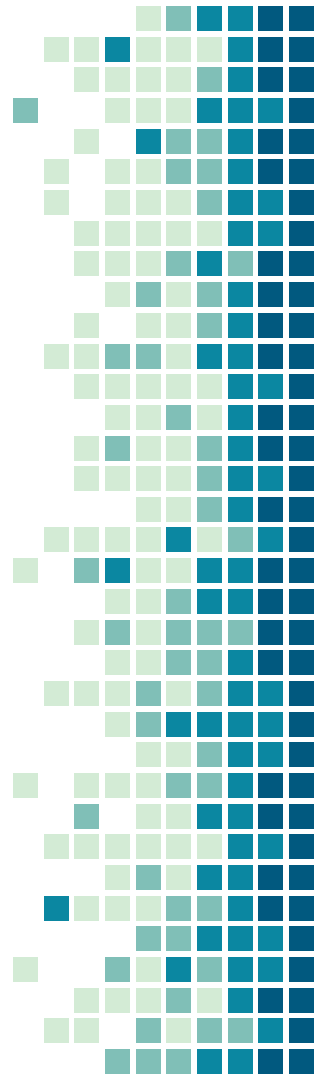
Optics and Displays

- Lens create distortion, in the (x,y) plane either as pincussion or barrel distortion
- Lens also cause chromatic aberrations depending on the color
- Corrections are made in software before sent to the display



Practical Use Cases

- Medical Training and Practice
- 3D medical scan analysis
- Technical Training
- Large complex data analysis
- Immersive Movies & Theme Park Rides
- Military Training
- Spectating Sports
- Tourism and Travel



Sources

- <http://www.realitytechnologies.com/augmented-reality>
- <http://www.wikitude.com/blog-shaping-future-technology-slam/>
- <http://www.sciencedirect.com/science/article/pii/S1877705812019698>
- <https://www.forbes.com/sites/quora/2017/01/09/how-do-augmented-reality-displays-work/#2cc085e47315>
- <http://www.ps-tech.com/3d-technology/optical-tracking>
- <https://github.com/ashtuchkin/vive-diy-position-sensor/>
- https://trmm.net/Lighthouse#HTC_Vive_Lighthouse
- <http://stanford.edu/class/ee267/lectures/lecture7.pdf>