Amazon Fulfillment Centers

Abdelwahab Bourai and Rohan Meringeti

Outline

- Introduction
- Motivation
- Background
- Physics
- Model 1: Linear Motion
- Model 2: Spherical Motion
- Model 3: Circular Motion
- Conclusions

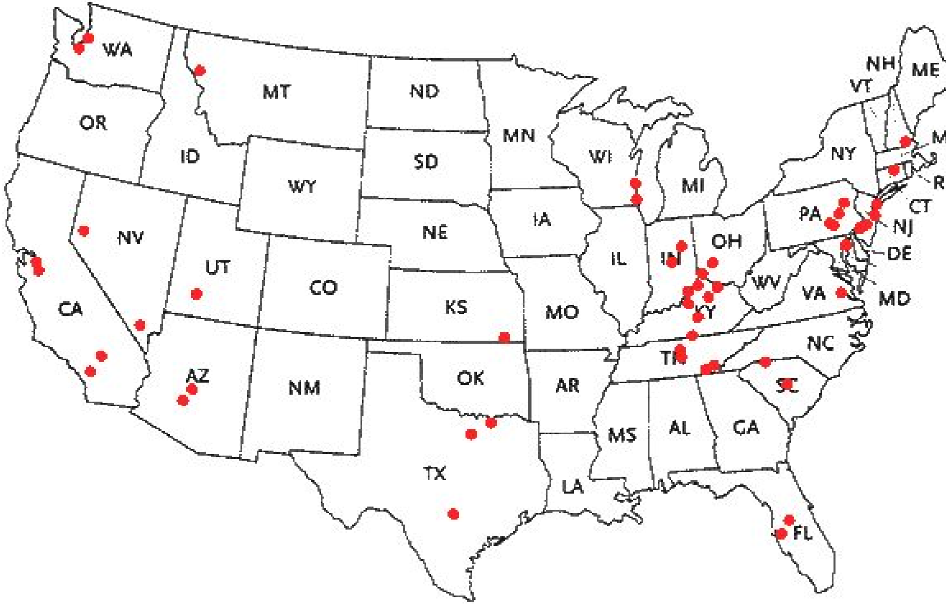
Introduction

$107 Billion

Dollars in sales from Amazon.com

20%

Growth in Revenue from 2014-2015



Fulfillment Centers distributed across the continental United States

Motivation for CPS

Why CPS?

- Robots constantly interacting with humans and other machines in warehouse
- Strong need for modeling of potential hazards and ways to safely and efficiently complete tasks
- Problem can be abstracted and applied in wide array of other applications such as autonomous vehicles, agricultural production, etc.

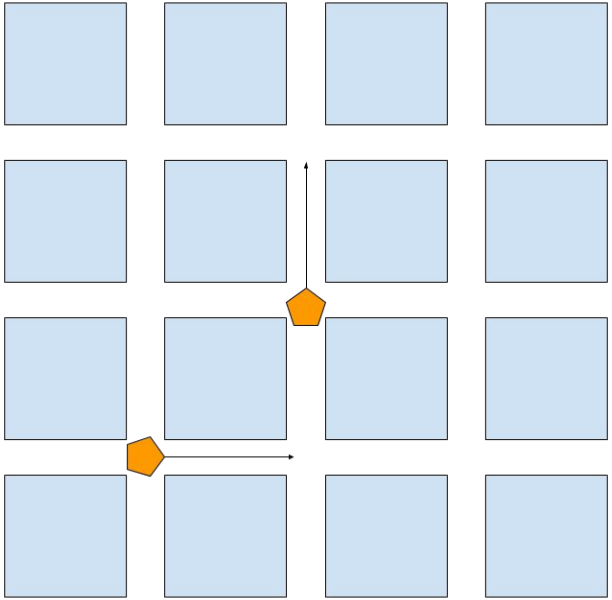
Challenges

Challenges

- Linear motion has been done in past and in our class, but what about constrained linear motion with robots attempting to complete tasks within close quarters?
- No precedent for modeling drone movements in 3D space, decided on using spherical motion
- How best to simplify a huge problem with many moving parts into three simple models?

Models

Linear Motion On a Grid



Linear Motion on a Grid (Travel along Grid)

```
/*Go in the direction (x or y) of the package which is the furthest distance from the robot*/
{
?(distToInterA = 0 & pkgPosxA > posxA & absHorA >= absVertA); {vertdirA:= 0; hordirA:=1; distToInterA := GSize;} ++  /* Go right*/
?(distToInterA = 0 & pkgPosxA < posxA & absHorA >= absVertA); {vertdirA:= 0; hordirA:=-1; distToInterA := GSize;} ++ /* Go left */
?(distToInterA = 0 & pkgPosyA > posyA & absVertA >= absHorA); {vertdirA:= 1; hordirA:=0; distToInterA := GSize; }++  /* Go up */
?(distToInterA = 0 & pkgPosyA < posyA & absVertA >= absHorA); {vertdirA:= -1;hordirA:=0; distToInterA := GSize;} ++  /* Go down */

?(distToInterB = 0 & pkgPosxB > posxB & absHorA >= absVertA); {vertdirB:= 0; hordirB:=1; distToInterB := GSize;} ++  /* Go right */
?(distToInterB = 0 & pkgPosxB < posxB & absHorA >= absVertA); {vertdirB:= 0; hordirB:=-1; distToInterB := GSize;} ++ /* Go left */
?(distToInterB = 0 & pkgPosyB > posyB & absVertB >= absHorB); {vertdirB:= 1; hordirB:=0; distToInterB := GSize;} ++   /* Go up */
?(distToInterB = 0 & pkgPosyB < posyB & absVertB >= absHorB); {vertdirB:= -1;hordirB:=0; distToInterB := GSize;}   /* Go down */

}
```

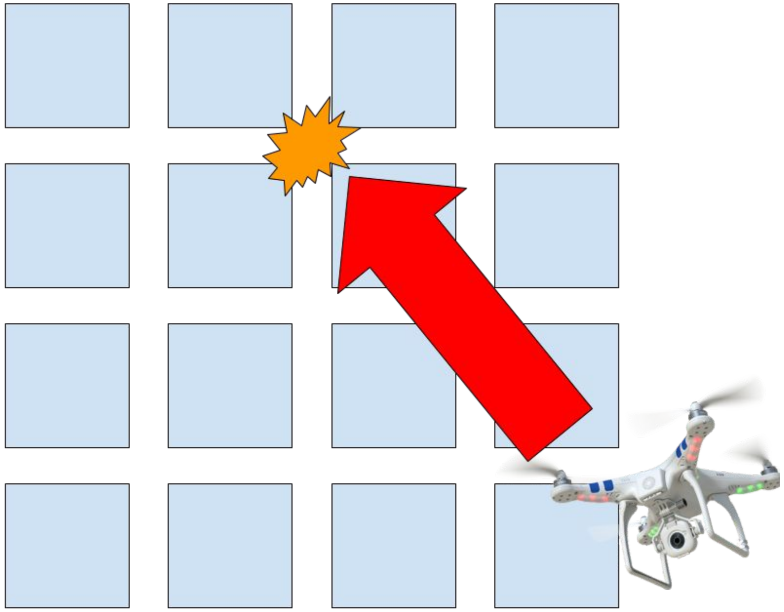
Linear Motion on a Grid (Locks)

```
}
{
  ?(((futureposxA -futureposxB)^2 + (futureposyA -futureposyB)^2)^(1/2)<= SafeRobotDist);
  {
  ?(OrderA < OrderB); hordirB:=hordirB*-1; vertdirB:=vertdirB*-1;} ++

  ?(((futureposxA -futureposxB)^2 + (futureposyA -futureposyB)^2)^(1/2)<= SafeRobotDist);
  ?(OrderB <     OrderA); hordirA:=hordirA*-1; vertdirA:=vertdirA*-1;
}
```

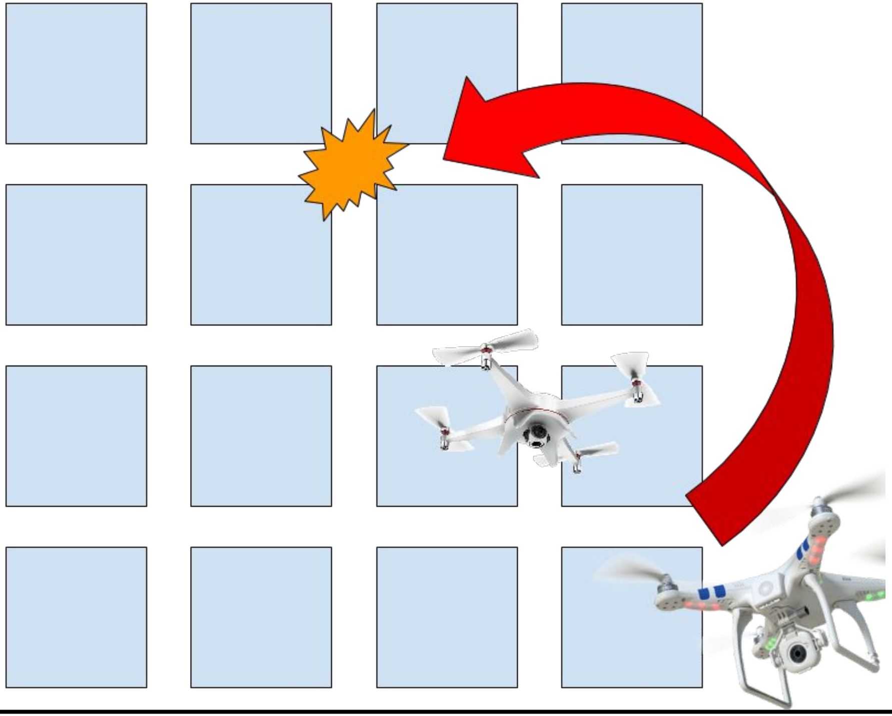
Linear Motion on a Grid (Continuous Dynamics)

```
  /* CONTINUOUS DYNAMICS */
  {
  posxA' = hordirA * velA, posyA' = vertdirA * velA, posxB' = hordirB *velB, posyB' = vertdirB * velB, velA' = 0, velB' =0 ,
  distToInterA' = -1 *(((velA)^2)^(1/2)), distToInterB' = -1 *(((velA)^2)^(1/2)) &  /* x direction ODE robot A */
     t' = 1
     & t <= T

  }

  }*@invariant(((posxA -posxB)^2 + (posyA -posyB)^2)^(1/2) > SafeRobotDist)
](((posxA -posxB)^2 + (posyA -posyB)^2)^(1/2) > SafeRobotDist) /* safety condition */
```

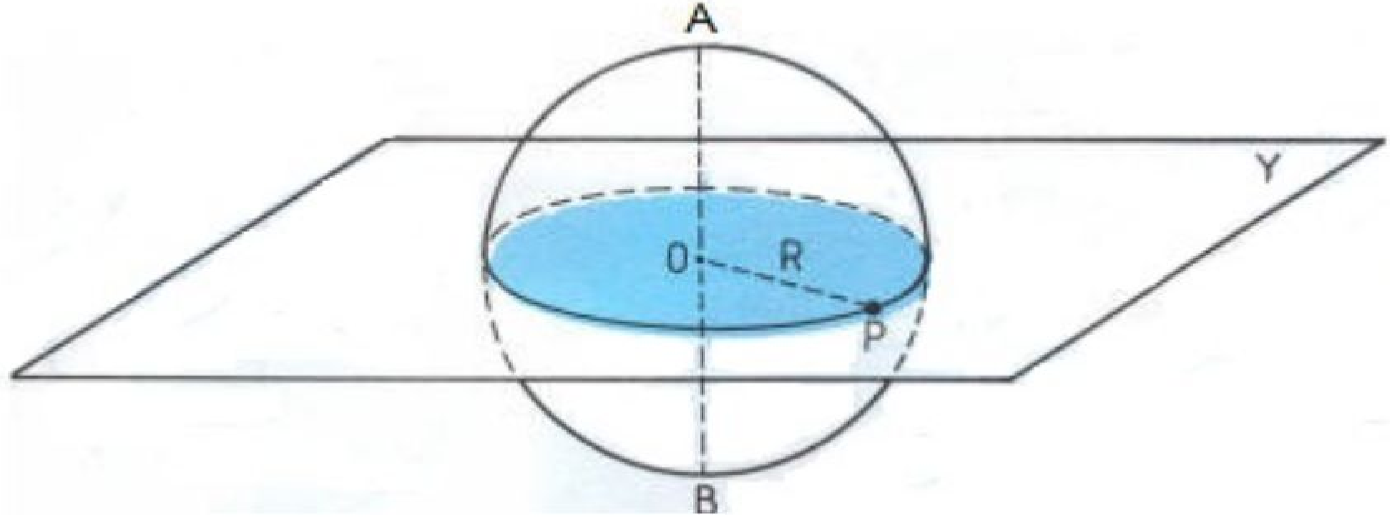
Spherical Drone Motion



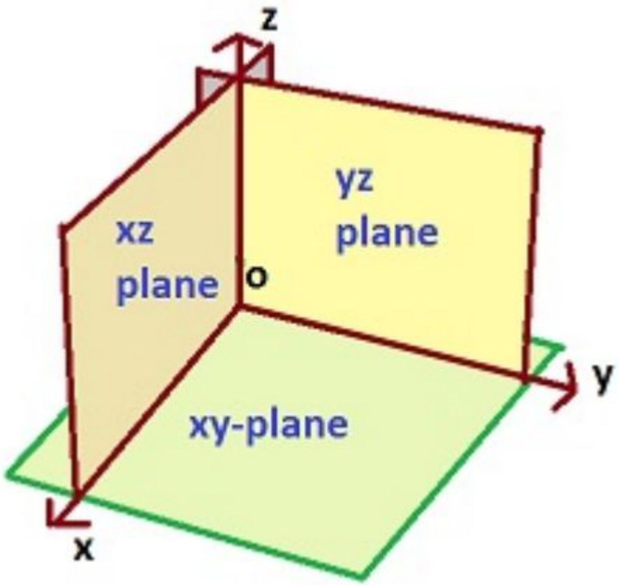
Spherical Drone Motion



Spherical Drone Motion



Spherical Drone Motion



Spherical Drone Motion (Linear)

```
obsDistance := ((obsX - x)^2 +(obsY - y)^2 + (obsZ -z)^2)^(1/2);

{
  ?(obsDistance > SafeRobotDist);

  magnitude := ((((emerX - x)^2) + ((emerY - y)^2) + ((emerZ - z)^2))^(1/2));

  unitvx := ((emerX - x)^2)^(1/2)/magnitude;
  unitvy := ((emerY - y)^2)^(1/2)/magnitude;
  unitvz := ((emerZ - z)^2)^(1/2)/magnitude;

  vx := MaxVelocity * unitvx;
  vy := MaxVelocity * unitvy;
  vz := MaxVelocity * unitvz;


  futureDist := ((obsX - (x + vx * T))^2 + (obsY - (y + vy * T))^2 + (obsZ - (z + vz * T))^2)^(1/2);
}
```

Spherical Drone Motion

```
/* This is what we do when we are on the sphere */

scaleV:=*;

planeX := x + unitvx * scaleV; /** Point on other side of the sphere **/
planeY := y + unitvy * scaleV;
planeZ := z + unitvz * scaleV;
?(((planeX - obsX)^2 + (planeY - obsY)^2 + (planeZ - obsZ)^2)^(1/2) = SafeRobotDist);

/** Use two vectors to define a plane **/

v1X:=((planeX - x)^2)^(1/2); /** vector from robot to other point on sphere **/
v1Y:=((planeY - y)^2)^(1/2);
v1Z:=((planeZ - z)^2)^(1/2);

v2X:=((planeX - obsX)^2)^(1/2); /** vector from obstacle to point on sphere **/
v2Y:=((planeY - obsY)^2)^(1/2);
v2Z:=((planeZ - obsZ)^2)^(1/2);
```

Spherical Drone Motion

```
/** Create normal vector to plane **/

normalX:=(v1Y * v2Z - v1Z * v2Y);
normalY:=(v1Z * v2X - v1X * v2Z);
normalZ:=(v1X * v2Y - v1Y * v2X);

normalMagnitude:=((normalX)^2 + (normalY)^2  +(normalZ)^2)^(1/2);
normalUnitX:=normalX/normalMagnitude;
normalUnitY:=normalY/normalMagnitude;
normalUnitZ:=normalZ/normalMagnitude;

/** create normal vector each coordinate plane **/

normalPlaneX:=1; ++ normalPlaneX:=-1;
normalPlaneY:=1; ++ normalPlaneY:=-1;
normalPlaneZ:=1; ++ normalPlaneZ:=-1;
```

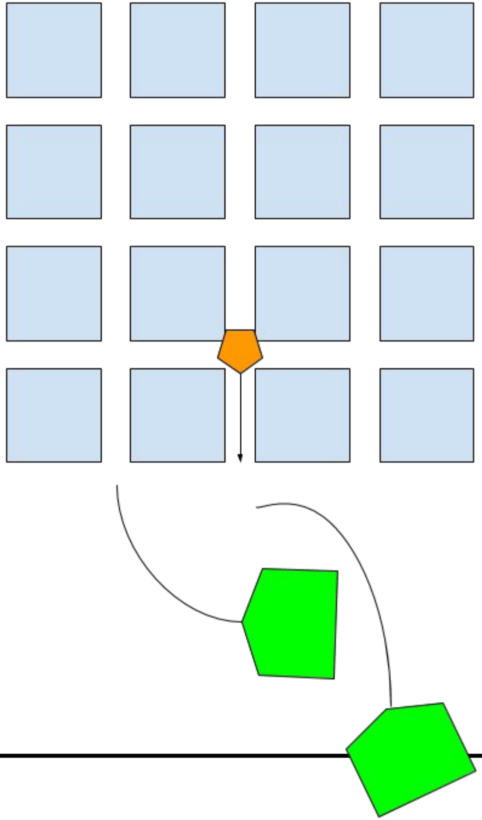
Spherical Drone Motion

```
/** Subtract the distance between two unit vector points from the Max distance so we can later
** scale how close the plane is to a particular coordinate plane. For example, if our plane was on the
** xy plane, our xy_distance would be 0. Subtracting that from maxAngleDistance would allow us to later
** scale our values so that we know that our entire plane is on the xy plane.
**/
xydist := MaxAngleDistance - (normalUnitX^2 + normalUnitY^2 + (normalUnitZ - normalPlaneZ)^2)^(1/2);
yzdist := MaxAngleDistance - ((normalUnitX - normalPlaneX)^2 + normalUnitY^2 + normalUnitZ^2)^(1/2);
xzdist := MaxAngleDistance - (normalUnitX^2 + (normalUnitY - normalPlaneY)^2 + normalUnitZ^2)^(1/2);

/** We only want the acute angles. If we get the other angle, switch the normalplane vector to get the
** other one
**/

?(xydist > 0 & yzdist  > 0 & xzdist > 0 );

/** Scale the distances so that they add up to 1 **/

vScale :=*;

xyweight := xydist/vScale;
yzweight := yzdist/vScale;
xzweight := xzdist/vScale;

?(xyweight + yzweight + xzweight =1);
```

Spherical Drone Motion(Continuous Dynamics)

```
**/
?(futureDist = SafeRobotDist);
{
  x' = MaxVelocity*dx,
  y'= MaxVelocity*dy,
  z' = MaxVelocity*dz,
  /** direction * (plane weight*MaxVelocity/ radius) **/

dx' = (-dy * xyweight*MaxVelocity/SafeRobotDist) +  (-dz* xzweight*MaxVelocity /SafeRobotDist),


dy' = (dx * xyweight*MaxVelocity /SafeRobotDist) +  (-dz * yzweight*MaxVelocity /SafeRobotDist)


dz' = (dx * xzweight*MaxVelocity /SafeRobotDist) +  (dy * yzweight*MaxVelocity /SafeRobotDist),

t'=1 & t < T
}
```

Circular Motion



Circular Motion (Handoff)

```
{?(((obsPosX-posxB)^2 + (obsPosY-posyB)^2)^(1/2)>SafeRobotDist & pkgPickupX=-1 & pkgPickupY=-1);
{
  posxA' = hordirA * velA,
  posyA' = vertdirA * velA,
  posxB'= 0,
  posyB' = 0,
  t' = 1
  & t <= T

}} /** If the circular robot is a safe distance away and the package has been dropped off, we head towards the
package **/
++
{?((obsPosX-posxB)^2 + (obsPosY-posyB)^2)^(1/2) > SafeRobotDist;
{
  posxB'=velB* hordirB,
  posyB' = velB *vertdirB,
  velA' = 0,
  t' = 1
  & t <= T
}}
```

Conclusions

- We have shown how to distill a complex problem into three separate models that interact with one another
- Distributed robotics will play an integral role in advancing Amazon’s goal in becoming the #1 retailer in the world
- Wide array of applications will find this system useful, from farming to retail stores