Amazon Fulfillment Centers: Robot Interactions in a Distributed System Paradigm

Abdelwahab Bourai, Rohan Meringeti

May 2, 2016

Abstract

In today's world, robots play an ever increasing role in tasks ranging from surgical procedures to manufacturing. In addition, as costs of labor continue to soar, the need for cost efficient alternatives has also risen. This need is often met through automation and robotic systems. However, this raises concerns of safety and efficiency in the workplace. In some industries, such as car manufacturing, the marriage of robots and labor has far surpassed expectations. However, newer industries are still experimenting with robotic involvement in the office. In particular, we investigate Amazon fulfillment centers and the emergence of automated warehousing systems, where robots carry out labor intensive tasks such as sorting and delivering merchandise throughout the facility. We model multiple robots interacting in a diverse set of scenarios, ranging from units that focus on finding and delivering packages in an organized environment to drones that respond to emergencies in the facility.

0.1 Introduction

Since its founding in 1994, Amazon has grown into the premier online marketplace with over 244 million active users and 2 billion items purchased per year. In order to ensure optimal customer satisfaction, the company has to ship orders at a breakneck pace. To do this, Amazon has set up 89 warehouses distributed throughout the United States known as fulfillment centers A robotics company called Kiva Systems was acquired in 2012 to explore the use of machines in product retrieval inside the warehouse. They are controlled by a central computer and an array of sensors, allowing for a distributed approach to the difficult problem of package delivery. The robots are constantly interacting with humans, as they deliver product shelves to workers for sorting and product selection. This symbiotic relationship between human and machine is vital to maximize efficiency in seemingly impossibly large tasks. However, this also raises the bar for safety and hazard prevention. We believe cyber-physical systems research can be applied in this field to model common scenarios and prove safety to minimize the risks and identify potential pain points in large distributed robot systems. Amazon net a profit of 89 billion dollars and 107 billion dollars in sales in 2014. In addition it experienced a 20% growth rate in revenue from 2014 to 2015. It can further expect to increase profits as it moves towards larger systems of distributed robots that can drastically cut labor costs throughout its fulfillment centers. Furthermore, these systems are not limited just to this specific scenario; distributed robotics systems can play an enormous role in industries and scenarios such disaster rescue efforts, agriculture, and autonomous driving systems.

0.2 Models

0.2.1 Model Background

We wish to model the system in a manner that simplifies the complexities of movement in the facility and also generalizes well. We divided the warehouse facility into three regions, each with its own main occupant. One region was where the human workers stand and wait for robots to deliver them products. Let this region be in the top third of the facility. In the middle third, we have a product shelving system organized into a grid, with "streets" and robots picking up products to deliver to the human workers. Finally we designate the bottom third as the "loading dock" for our sytem, where robots take recently arrived items and deliver them to the edge of the product shelves to be organized. Given this background, we identified three key models: a linear model focused on robots tasked with retrieving products from a grid system of shelves, a circular motion model to model robots tasked with delivering products recently arrived to the facility to the linear motion robots to be added to the grid, and a spherical motion robot model simulating drones that react and repair emergencies in the system. We model the individual robots as well as their interactions with each

other. In some models we have two sets of variables such as $posxA$ and $posxB$. Assume without loss of generality that when we mention variables denoted with A the same holds true for B. In addition, assume the robots we use in the system can move in any direction, meaning the robot can change direction without needing to turn.

0.2.2 Exploring Linear Grid Motion with Robots

We model two robots on a linear grid system. Both are completing the same tasks and travel within the same grid. We model movement throughout the grid by assuming the robot can only go in the vertical and horizontal planes; there is no diagonal movement. When we wish to pick up a package, or deliver a package to a certain location, we observe where the package is in the xy plane. To get to that point, we measure how far away we are in the vertical and horizontal directions and take the maximum of the two distances. In the model the variables $absVertA$ and $absHorA$ are compared. The distance that is greater determines which direction we travel in for the current time step.

$$absVertA := \sqrt{(pkgPosyA - posyA)^2}$$

$$absHorB := \sqrt{(pkgPosxA - posxA)^2}$$

$$absVertB := \sqrt{(pkgPosyB - posyB)^2}$$

$$absHorB := \sqrt{(pkgPosxB - posxB)^2}$$

Notice that we are taking the square root of a square to get the absolute value of the distances.

So now we are able to choose which direction we move in. However what if we need to switch directions? This requires us to model turning. We set up a variable $distToInterA$ that measures how far we are from the next intersection in our path. In our continuous dynamics, we decrement it based our velocity as we approach

$$distToInterA' = -1 \times \sqrt{(velA)^2}$$

Throughout this model, the robots are usually moving at speed $MaxVelocity$. However, if we detect that we are approaching an intersection and will overshoot, we adjust our velocity such that we will stop exactly at the intersection:

$$?(distToInterA > 0 \wedge distToInterA \leq MaxVelocity * T); velA := \frac{distToInterA}{T}$$

Now that we have stopped at the intersection we have to decide which direction to continue on. We previously mentioned how we always move in the direction that is furthest away from the package. We introduce two variables $horDirA$ and $vertDirA$ that act as flags denoting directions left, right, up and down. They have three possible values: 0, 1, and -1, with 0 implying not moving in that direction at all, 1 implying right and up, and -1 implying left and down.

So for example if our package is farther away to the right we would update our values this way:

$$?(distToInterA = 0 \wedge pkgPosxA > posxA \wedge absHorA \geq absVertA);$$

$$vertdirA := 0; hordirA := 1; distToInterA := GSize;$$

We reset $distToInterA$ to the maximum value of $GSize$, the length of one grid block.

Another issue we model is interacting with other robots on the grid. We always attempt to maintain a safe distance, set at $SafeDistance$ in the model, such that in the next time step we are certain of avoiding collisions. This distance is determined by taking the hypotenuse of the grid blocks, this way we ensure that no two robots will ever be close enough that they can turn into the same path and crash. In the case where two robots are very close in the grid, we set up an ordering system where the robot with the higher ordering retreats and heads in the opposite direction until both robots are at a safe distance apart. We determine where each robot will be in the next time step:

$$futureposxA := posxA + (velA * hordirA * T)$$

$$futureposyA := posyA + (velA * vertdirA * T)$$

$$futureposxB := posxB + (velB * hordirB * T)$$

$$futureposyB := posyB + (velB * vertdirB * T)$$

We use the previously mentioned direction flags to determine exactly where the robots will be on the grid after time step T. This cooperation between robots is vital for safely moving throughout the grid.

We made a few other key simplifications. We removed acceleration and braking from the system because we assume velocity to be negligible. Additionally we assume no two way streets in the grid. We deemed this to be too much of a safety risk with robots carrying fragile cargo so close to one another. While there may be a potential efficiency loss due to forcing robots to wait for paths to clear off, the safety gained from not having robots side by side outweighs those losses. We also make the assumption that each robot is at least a block's hypotenuse distance, $SafeDistance$, away from each other and set this as a precondition. This allows to never have to worry about two robots crashing into each other.

0.2.3 Spherical Motion for Emergency Response

The type of situation we attempt to model is akin to a an accident or emergency occurring. Since the goal is to resolve the emergency in the most efficient manner and packages are often height dependent (stored on high shelves), we model

a drone going towards a package, even with the obstruction of an obstacle along the way. The emergency response model is far more complex than the linear model. With the linear model, we operate only on the cartesian plane. However if we want to model drone movement and behavior we have to able to account for maximal range of motion for the drone to be effective in dealing with any type of emergency.

An obvious challenge in this model is how to efficiently bypass models to reach the emergency location. The idea we used here is that a drone approaches linearly, in a straight line, towards the emergency location. At each time step, we create a sphere of length $SafeRobotDist$ around our obstacle. The goal is make sure that we never get inside this sphere. The manner in which we conducted this test is take a unit vector in our current direction, and see the furthest we would go if the differential equation were to run for T time steps. If we were to go inside the sphere (have a euclidean distance of less than $SafeRobotDist$) we would slow down so at most we would end up at the surface of the sphere. Once on the sphere, we would chart a route on the surface of the sphere towards the emergency, and travel along that route to bypass the obstacle. We will elaborate on how this step is done in the following expansion of the model.

We now take a closer look a the model, and explain some of the challenges and decisions we made. Let us declare three groups of positional variables representing the drone, the hazard, and a potential obstacle:

$$x, y, z$$

$$emerX, emerY, emerZ$$

$$obsX, obsY, obsZ$$

The hazard in our instance can be a stalled robot, a spill, or a crash. The obstacle could be a human worker in the facility other drones, or any inanimate object in the way. We need to reach the emergency as quickly as possible, so the first step is to see if we can travel directly to the point using linear motion. We take the Euclidean distance between us and the obstacle

$$obsDistance := \sqrt{(obsX - x)^2 + (obsY - y)^2 + (obsZ - z)^2}$$

and ensure that it is greater than $SafeDistance$. If it is we continue on a straight path to the hazard. We then find the magnitude of the line from our drone to the $emerX, emerY$ and $emerZ$. We do this to get the unit vectors:

$$unitvx := \frac{\sqrt{(emerX - x)^2}}{magnitude}$$

$$unitvy := \frac{\sqrt{(emerY - y)^2}}{magnitude}$$

$$unitvz := \frac{\sqrt{(emerZ - z)^2}}{magnitude}$$

With these we can know what our velocities will be in the x, y and z directions.

$$vx := MaxVelocity * unitvx$$

$$vy := MaxVelocity * unitvy$$

$$vz := MaxVelocity * unitvz$$

It's very unlikely we can continue with linear motion forever, so we calculate how much ground we can cover with linear motion until we have to avoid some obstacles. This is stored in $futureDist$ in the model

$$futureDist := \sqrt{(obsX - (x + vx * T))^2 + (obsY - (y + vy * T))^2 + (obsZ - (z + vz * T))^2}$$

Now that we have reached a point where linear motion no longer suffices, we must introduce spherical motion. We are now traveling on a sphere of radius $SafeDistance$ with the obstacle at the center. Our goal is travel along the sphere such that we approach our emergency. To do this we intersect a plane to the sphere like below. One caveat with this approach however, is that we must deal with the scenario where we are outside of the sphere starting out, and in the next time step, we are also out of the sphere. One possible path taken is right through the sphere, which would break a safety condition that states that we must remain outside of the radius of the sphere. Fortunately, the very fact that we're cutting through the sphere in one time step means that we never encounter the obstacle, since we have the precondition, $MaxVelocity <= SafeRobotDist/T$, which guarantees, that we can travel a max distance of SafeRobotDist in one time step. Therefore, we have adjusted our safety condition to that the obstacle and the drone never hit.

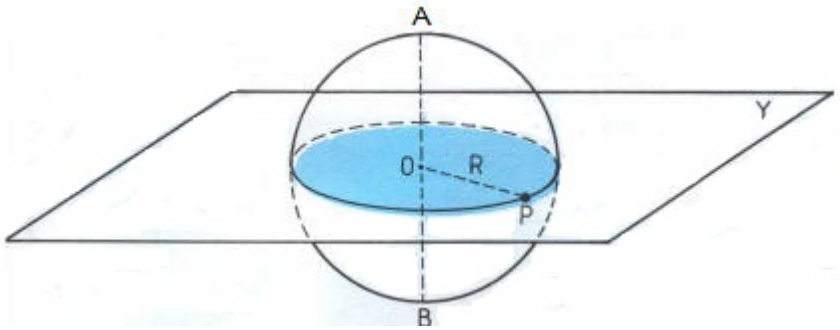

Figure 1: Intersection of a Plane and Sphere

We want to find planes that intersect with the sphere on the all three coordinates. To define a plane, we need three points. Our first two points are easy,

they are the obstacle, and position of the robot themselves. The final point is gotten by taking a line from the robot to the emergency point. The point on the other side of the sphere is the third point of the plane. This is found through non deterministic assignment, until we have reached a SafeRobotDist.

$$scaleV := *$$

$$planeX := x + unitvx * scaleV$$

$$planeY := y + unitvy * scaleV$$

$$planeZ := z + unitvz * scaleV$$

Those unit vectors we made earlier prove useful, as they tell us which direction we go in each coordinate plane. Given these points, we have to test to make sure we will actually be safe once we reach there, if not we choose a new point on the sphere

$$?(\sqrt{(planeX - obsX)^2 + (planeY - obsY)^2 + (planeZ - obsZ)^2} = SafeRobotDist)$$

So taking a look at where we have reached till now, we have the equation of plane intersecting with a sphere, which represents the route we wish to take. The next part of spherical motion takes into account the idea of pairwise planes. To understand how pairwise planes work, we have to take a look at circular motion.

Circular motion is done within one plane, generally on the coordinate planes (xy, yz, or xz). The differential equations used in circular motion are the following.

$$x' = v * dx$$

$$y' = v * dy$$

$$dx' = -dy * v/radius$$

$$dy' = dx * v/radius$$

.

The point to highlight here, is value $v/radius$ which tells us the rate that the angle is changing around the circle. The idea we now use is one with pairwise planes. Suppose we have,

$$x' = v * dx, y' = v * dy, dx' = -dy * v1/radius, dy' = dx * v1/radius$$

.

$$x' = v * dx, z' = v * dz, dx' = -dz * v2/radius, dz' = dx * v2/radius$$

.

$$y' = v * dy, z' = v * dz, dy' = -dz * v3/radius, dz' = dy * v3/radius$$

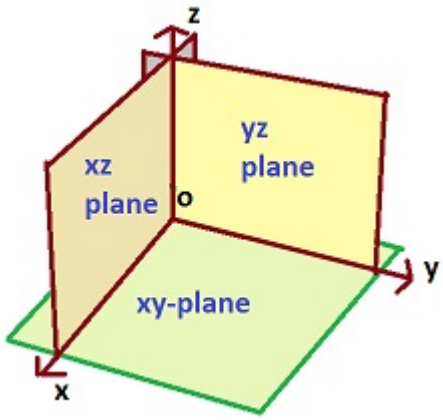

Figure 2: Pairwise Coordinate Axes

.

So this is essentially circular motion on each coordinate plane. So how can we relate this idea to the plane we have created? We see that the only difference between the planes above the rate of change of angles. We also know that that any plane is a weighted average of the 3 pairwise planes above. In other words, we have a unit velocity vector in the direction of our plane, but how do we decompose that velocity vector in to the three coordinate planes, such that we get circular movement on that plane. Essentially, this question boils down to how the velocity vectors in each direction are weighted. The tactic that we use to solve this is the idea of plane angle similarity. We take the angle between our plane and each one of the coordinate planes, and weight the rate of change of angle in that direction based on how close that angle is with the coordinate plane. For a simple example, say our plane was actually the xy-coordinate plane. Then the angle difference with the xy coordinate plane would be 0 degrees, while there would be a 90 degree difference with the other two planes. We would thus gives the v_xy plane a weight of 1, and the other two 0 weights, leaving only the circular motion in the xy coordinate plane. The code below shows how these weights are gotten and applied.

$$normalX := (v1Y * v2Z - v1Z * v2Y);$$

$$normalY := (v1Z * v2X - v1X * v2Z);$$

$$normalZ := (v1X * v2Y - v1Y * v2X);$$

$$normalMagnitude := ((normalX)^2 + (normalY)^2 + (normalZ)^2)^(1/2);$$

$$normalUnitX := normalX/normalMagnitude;$$

$$normalUnitY := normalY/normalMagnitude;$$

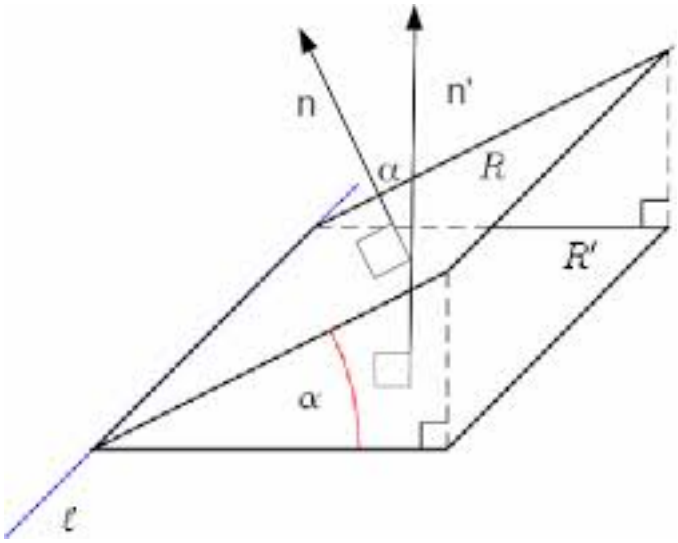

Figure 3: Angles between Planes

$$normalUnitZ := normalZ/normalMagnitude;$$

$$normalPlaneX := 1; ++ normalPlaneX := -1;$$

$$normalPlaneY := 1; ++ normalPlaneY := -1;$$

$$normalPlaneZ := 1; ++ normalPlaneZ := -1;$$

$$xydist := MaxAngleDistance - (normalUnitX^2 + normalUnitY^2 + (normalUnitZ - normalPlaneZ)^2)^{1/2};$$

$$yzdist := MaxAngleDistance - ((normalUnitX - normalPlaneX)^2 + normalUnitY^2 + normalUnitZ^2)^{1/2};$$

$$xzdist := MaxAngleDistance - (normalUnitX^2 + (normalUnitY - normalPlaneY)^2 + normalUnitZ^2)^{1/2};$$

Notice how we did mention that we would find the angles between the planes, however, there are no angles used in the above equations. This is due to two reasons. One is that the equation for angle between two planes involves trignometric calculations, which is not possible in Keymeara. Second, is that we don't really care about the exact angles. Instead what we care about is the ratio of angles to each other. Therefore, what we do is we take a normal unit vector, one from our plane, and one from each coordinate plane, and compare the distance between them. We only want the distance formed by an acute angle, therefore the max distance between any two points is simply $\sqrt{1^2 + 1^2} = \sqrt{2}$. For example say we compare the normal vectors of our plane and the xy coordinate plane. If our plane were to be the xy coordinate plane, we would have a distance of 0. On the otherhand, say our plane was the yz, or the xz plane, then we would have a distance of $\sqrt{2}$. The choice used in the normalPlane variables above is so that

we only get the actue angles, and the test $?(xydist > 0 yzdist > 0 xzdist > 0)$; makes sure of this fact. We subtract each distance from $\sqrt{2}$ or MaxAngleDistance, so that we can get a ratio of the similarity of the angles, not differences. Finally, in the following lines, we scale the weights appropriately, so that the sum equals 1.

$$vScale := *;$$
$$xyweight := xydist/vScale;$$
$$yzweight := yzdist/vScale;$$
$$xzweight := xzdist/vScale;$$
$$?(xyweight + yzweight + xzweight = 1);$$

0.2.4 Interaction between Linear and Circular Motion Robots

The final model is based on the scenario where a robot takes recently delivered, unsorted merchandise to the boundary of the product grid for the linear motion robots to pick up and place in their proper places. Our preconditions are similar to those of the linear model; we do not want to start at an unsafe distance from the other robot. However another precondition is

$$posyB < LoadingBoundary$$

meaning that robot B, our circular motion robot, must be below $LoadingBoundary$, which is set as the y value for the border between the loading region and the product grid. We let our Robot A run as in model 1, but this time without having to worry about another linear motion robot, thus its motions are quite simple. The interesting portion of this model is how the two robots interact with each other. For example, what is robot B doing while robot A is dropping off a package. One option is that is travels underneath the loading boundary looking for packages. Ideally, however, the locations of where packages are dropped off are handled by the same system that controls the robots. Therefore, the robot really shouldn't move until the system tells it to grab a package in the loading boundary. Hence, we introduce the following variables to alert the circular robot to where it should go.

$$pkgPickupX$$
$$pkgPickupY$$
$$pkgDropoffY$$

Furthermore, we don't want the model to be too boring, so we add an obstacle in the path of our circular robot. Similarly to how we handle the sphere situation, we attempt linear motion towards the package when we are at a safe distance away, and circular motion when we are enroute to hit the obstacle. We have already mentioned much of circular motion in the previous model, so we won't elaborate on the concept here.

0.3 Proofs

0.3.1 Proof 1

The grunt of this model, involves a lot of cases and choice statements. Therefore, the strategy used to prove this model is loop induction, followed by breaking on choice and composition. Once each branch is broken to its child nodes, we can solve each respective branch through Differential Induction, which captures the idea that distance between the two robots will never be closer than SafeRobot-Dist.

0.3.2 Proof 2

0.3.3 Proof 3

0.4 Applications

0.4.1 Agricultural Labor Improvement

Distributed robotics systems are capable of taking over simple, labor intensive tasks that allow humans to focus on more complex tasks. For example, crop picking in California is an enormous industry and generates billions of dollars in agricultural revenue. In Ventura County alone, 20,000 jobs are added every crop cycle for fruit picking. However, many of these farms are still desperate for more workers, leading some to turn to hiring practices that are both dangerous and illegal. A system set up similar to our model can be perfect for not only picking, but also optimizing picking production. Imagine an orange orchard in the Central Valley in prime picking season, with thousands of acres of oranges read to be picked. We propose a few different possibilities for our system.

The first one focuses on having the simplest possible system. Robots are spread throughout the orchard, with each robot operating in a small square plot of trees. Human workers pick the oranges and place them in bins, and when they are full the robot in charge of that plot collects and drops off these bins at a central pickup location. Another class of robots waits at that boundary and delivers it to the large sorting bins for human quality control workers to pick out rotten crops and package them for delivery. Since we have worked on obstacle avoidance and working side by side human workers, the distributed robot model we proposed above for both the linear grid motion and the interactions with the circular motion model can be replicated easily here.

Another, more complex task involves using our spherical motion model in direct crop picking. If instead of heading to an emergency we tune our drone model to identify crops and outfit it with the proper manipulators, a large swarm of drones, working in tandem with a distributed system of ground robots that aid in the transport of the picked fruit, can do the grueling work in the hot Central

California day and allow human workers to once again focus on auxiliary tasks. This can be a boon for labor rights, as it will allow farmer currently skirting labor laws to deal with the huge labor demand to pick crops. If they can minimize the amount of workers needed, they can pay at or above the minimum wage, rather than below for a full days of work in the hot sun. In addition, these robot systems would pay themselves off in the long run compared to the current system, as the huge influx of thousands of workers during crop picking months places strain on local municipalities.

0.4.2 Vehicle Automation

Research in self-driving vehicles has accelerated in recent years, as more companies throw their hat into the ring in the hopes of developing safe autonomous cars. Already some cars offer automated brake assistance or in the case of Tesla autopilot features. One company that may have a particularly interesting system will be Uber, which is trying to replace human drivers with driverless cars. The cars will be available to consumers at the click of a button, but there is potential for cooperation and interaction between the vehicles. Imagine the "cargo" for our model being the humans scattered across a city grid. Our linear motion model would have to be expanded to deal with two way streets and much more robust safety concerns. In addition, we can utilize our linear and circular interaction model to deal with problems such as freeway driving.

For example, in New York City, and most cities, most of the city is laid out in a grid street system, with very few non-linear motion paths. However, if someone were to request a trip to the suburbs, or vice versa, there may be a need to take a freeway, a much more complex task that involves far more non-linear motion. A distributed robot system can identify these issues and have two modes of transportation. When in the city, and requesting destinations in the city, the system only calls grid-based robotic cars. However, if one wishes to get on the freeway, grid motion vehicles can pick up the passenger and take them to a "loading dock" of sorts where longer distance and specialized freeway cars await to transport them out of the city to areas with less developed city planning.

0.5 Conclusion

The application of cyberphysical systems on to interactions between robots is endless. By proving that robots can work together to solve problems and handle any emergencies, the need for human supervision is redacted. In our situations modelled in this paper, we attempt to understand and explain the processes used in Amazon Fulfillment Centers, where multiply classes of robots must work together to accomplish a goal. In such an environment, having humans on the

work floor is an unnecessary hazard, so it is critical that these robots work together safely and efficiently without any constant supervision.

We have modeled three scenarios, which we believe are critical to the functioning of the warehouse. The first is through the use of linear motion robots whose main function it is to pickup and dropoff packages to a LoadingBoundary. This scenario is critical to the shipping and handling process of the warehouse, and involves multiple robots working a "city-grid" like format. These robots have to deal with issues such as turning, coordination through the use of locks, and determining correct paths to take to obstruct the least number of robots. Particularly the last problem is the hardest to solve, as difficulty of the algorithm increases exponentially for each new robot added.

The second scenario is the most cutting edge in technology and uses drones to fix emergency locations, and solves any problems on the work floor. The use of the drones is invaluable, since no matter how reliable CPS modeling and tests may be, reality is always different. Furthermore, if we had humans fix the emergency, we would have to halt production on the entire grid to make sure the safety of the worker isn't compromised. This is akin to shutting down an entire city to fix a road. If instead drones were able to fix these issues, both productivity and safety issues would be resolved.

Finally, the third scenario involves the coordination between two different classes of robots. We modeled the interactions between a linear robot, which delivers a package from the warehouse to the loading boundary, where it is then picked up by a circular robot, which may have a task such as loading on a specific truck for shipment. The model is crucial since it represents how different robots with completely separate abilities are able to work together to complete a goal.

With these scenarios in mind, we modeled basic functionality of an Amazon Warehouse. We were able to prove both the linear motion robots and the coordination between the linear and circular motion robots which showed us that robots can cooperate safely in a system to achieve a common goal. Although we were not able to prove the spherical drone model, this is mostly due to the edge case of going through a non center point of the sphere within one time step. Given the right pre-conditions and loop invariants even this condition can be met and proved. In conclusion, we have developed the basic dynamics for understanding how a complex cyberphysical system can be modeled and used to guarantee safety of different realms of robots working in tandem to accomplish a singular goal.

Bibliography

Todo