Modeling of K-Pop Dance Choreography as a Synchronized Multi-Agent Hybrid System

Anne He
afhe@andrew.cmu.edu

December 18, 2020

1 Abstract

This project aims to model and prove the motion of a synchronized multi-agent system, inspired by the formation and movement of dancers in K-Pop choreography. K-Pop often involves large groups of moving dancers that have to move to specific positions within a tight time constraint. Deciding the placement and movement of dancers such that they do not collide while changing positions is a challenging problem for choreographers, and is also a challenging problem within cyber-physical systems. Therefore, this project seeks to formally model and verify the safety of common group formations and understand the limitations on the timing and spacing of transitions between multiple agents. In this paper, I introduce and prove event-triggered controllers that model the behavior of small group transitions that can be extrapolated to larger groups, and propose bounds on the time required to successfully complete a transition based on the paths of the dancers. I successfully prove time and position bounds for a 1-D case and controllers for the 2-D case. I also prove the upper bound on unsafe transition time for multiple agents in 2-D, and propose a proof strategy for lower bounding safe transition time. This work is novel not only in its setting, but also because it includes specific time and position safety constraints that are not often considered when modeling standard cyber-physical systems. Likewise, the ideas discussed in this paper could potentially be generalized to other multi-agent problems regarding the simulation of the formation of birds, fish, bees, or other swarm applications.

2 Introduction

K-Pop is a global phenomena that has captured audiences with its elaborate and synchronized performances, often involving large groups of dancers. Coordinating formations that satisfy necessary constraints, such as being both feasible and visually appealing, is one of the most challenging parts of choreographing a piece. K-Pop choreography consists of dancers dancing in formation, which is the arrangement of dancers in a group. The placement of dancers in this formation is determined by many variables, such as being able to safely move around, or being visible to the audience. A dance consists of many different formations, and dancers also frequently swap places in this formation while keeping the same general shape of the group. A transition can be defined as either all the dancers moving to a new position to create a new formation, or a subset of dancers switching positions within the same formation. The complexity of the formations and transitions increases greater than exponentially in relation to the number of dancers. The choreographer must consider various properties for each individual dancer and ensure the safety of the transition. Therefore this project seeks to assist in this process or at least have a better understanding of it by modeling safe formations and transitions in choreography.

In this model, we treat each dancer as an individually autonomous robot within a system of multiple similar robots. We define a safe transition as one where all dancers have a safe path from point A, their original position, to point B, their target position. A safe path is defined as one where a dancer can move without colliding with other dancers, and the dancer must be able to reach the end point within a constant time T. The time T for a transition is usually defined by the music, therefore it establishes an upper bound on the travel time for all dancers. In order to maintain synchronization and because the music does not stop, it is imperative for all dancers to start moving at the same time and stop at their final positions safely within the time bounds. For this model, I restrict the proved model to a small number of agents and discuss how it can be extended to more agents.

The new challenges in this project compared to previous work include making sure the dancers reach a specific location and making sure that they do this within a certain time constraint. In previous labs and most existing CPS work, the safety of the position of the robot is only bounded such that the robot does not collide with obstacles. This project builds efficiency directly into the model. Additionally, this project takes advantage of the diamond notation used in dL because each dancer is responsible for choosing an optimal path to reach their goal, and models group synchronization in a unique way that allows efficient collision avoidance not present in other multi-agent systems where agents avoid collision by enforcing the same limits on each agent.

3 Related Work

There is much existing work on multi-agent systems and multi-agent motion planning, particularly regarding drone swarms and simulation of natural phenomena such as bees, birds, and fish. I noticed that many followed probability-based models or leader-follower models, and generally research focuses on the movement of the group as a whole and not movement within a group, which is the main focus of this project. Existing research largely aims to allow the group as a whole to avoid environmental obstacles. I read several papers from Dr. Magnus Egerstedt that had interesting insights on connected multi-agent systems and coordinating systems based on optimizing coverage or other properties between robots. Existing research seems to not place precise temporal and spacial bounds on robots. I did also find a paper proposing a temporal logic system for coordination of multi-agent systems. As I discovered while doing my project, without making certain assumptions about the motion of the robots, formally verifying precise temporal and spacial bounds can be very convoluted and generally is probably less of a priority in other situations. An interesting extension of my work would be to also include group motion and allow the robots to determine their positions within the group formation based on certain constraints such as visibility, distance from the centroid, and distance from other robots, but this is out of the scope of this class project.

4 Approach

First, I will discuss some assumptions used in these models. I assume that the dancers move within a 2D coordinate plane, both because dancers usually perform on a flat planar stage and because this works well with the limitations of KeYmaera X. In all of the models introduced in this project, each dancer starts at rest in a start position, and aims to reach a defined end position. Each dancer is aware of the position and velocity of the other dancers.

I chose to use an event-triggered model because the time at which a dancer reaches the position and the time at which a dancer changes velocity and direction is very crucial, and using a time-triggered model would introduce too many constraints to produce a reasonable simulation of how dancers move in real life. I also assume that dancers can change direction and velocity at any time within the system, with only a bound on the maximum velocity. In real life, the maximum velocities dancers travel at is slow enough such that one can change velocity and direction at any moment. While a person is walking in one direction, they can quickly step in another direction or take larger steps. Therefore, acceleration or time delay can be reasonably ignored in this scenario.

We will define each dancer as a finite point (x, y) and define a collision between dancers i and j as a point intersection where $(x_i, y_i) = (x_j, y_j)$. This implementation further supports ignoring acceleration because in real life, the dancers do not have to stop at an exact point location, only an approximate area around the location that is well within the ability of the dancer to stop in given the relatively slow maximum velocity.

Most transitions in K-Pop choreography do not necessarily involve all dancers moving and changing positions. Many of them will have a subset of nearby dancers switch positions so that a new dancer is at the center of the formation.

First let us observe some examples of how formations and transitions are laid out.

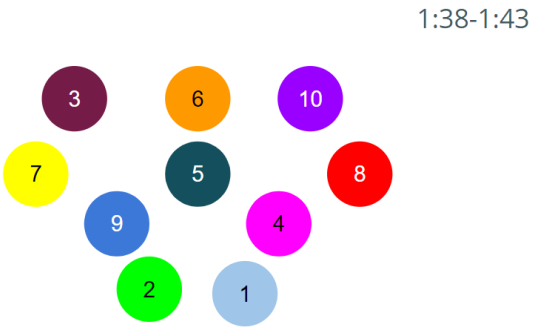

Figure 1: A diagram of a K-Pop dance formation used by choreographers. The top of the diagram represents the front of the stage. Each dancer is numbered and colored. The time interval in the corner roughly denotes the time that the group stays in this formation.

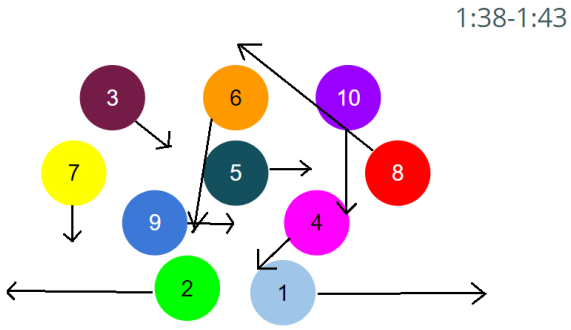

Figure 2: An example of a complex transition where all dancers shift to completely new positions in a new formation.

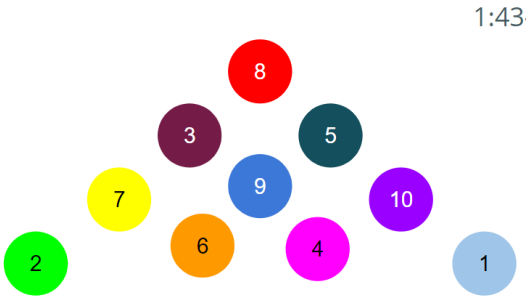

Figure 3: The next formation in the sequence following the transition in Figure 2. This represents one of the standard formations in K-Pop dance, a staggered triangle.

Transitions can be very complex with many dancers moving at once. Fortunately, these kinds of transitions are not very common. Formations often adhere to certain constraints that lead to nicely defined shapes, such as in Figure 3. Now we will observe a common type of transition from Figure 3.

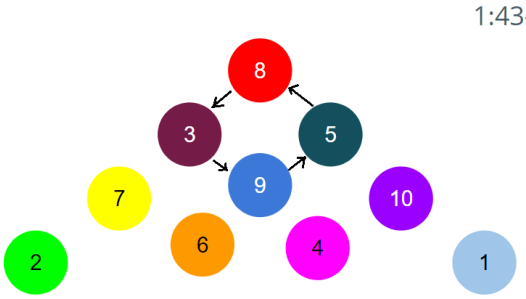

Figure 4: The next transition from the formation above. The four dancers in the diamond at the front rotate positions counterclockwise.

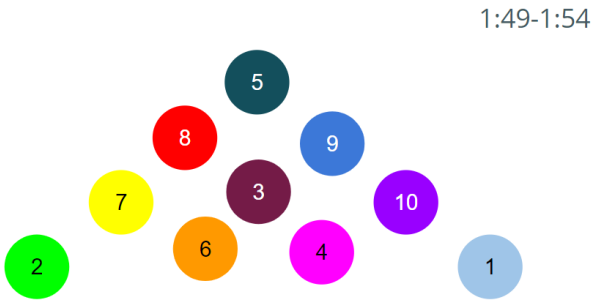

Figure 5: The result of following the transition above.

4.1 1D case - Time and Position Constrained

Now let us consider the most basic case, where a dancer is moving in a straight line with no obstacles or other dancers that could intersect their path. We can consider this as a 1-D problem. This situation is quite common, as one can see from Figure 4, where each dancer travels in a linear direction towards their new position. Of course, with multiple dancers, this introduces the possibility of collisions, which will be addressed in the 2-D case. For now, the challenge of the 1-D case is to identify and verify some time T such that a dancer cannot reach a defined point in any time $t < T$, and can reach a defined point in any time $t >= T$.

First let us address the upper time bound T on a dancer not being able to reach the target position, or the $t < T$ case. We can model this using box notation to state that, for all reachable states, the dancer robot cannot reach the target position. I limit the robot to traveling in the direction of the target, as traveling backwards would only further prevent the robot from possibly converging to the target, and because there are no obstacles there is no reason for the robot to change its direction.

Model 1 Robot in 1-D Cannot Reach in Time T

```
Definitions
  Real A;  /* Start position */
  Real B;  /* Target position */
  Real vmax; /* Maximum velocity */
  Real T; /* Maximum time for transition */
End.

ProgramVariables
  Real pos; /* Position */
  Real v; /* Velocity */
  Real t; /* Time */
End.

Problem
  (
     t=0 & v=0 & pos=A & T>0 & vmax>0 & A<B & T<(B-A)/vmax
  ) /* Initial conditions */
  ->
  [
    {
      /* Controller */
      {v:=*; ?(v>=0 & v<=vmax);} /* Assign velocity */
      {pos'=v, t'=1 & t<=T} /* Differential Equations */
```

```
    }*@invariant(t<=T & pos<=A+(t*vmax)) /* Loop Invariant */
  ] (pos!=B) /* Robot reaches end position */
End.
```

The intuition behind the proof of this model is fairly straightforward. From basic kinematics, we know that given max velocity $vmax$, the dancer cannot get from point A to point B in less than $\frac{A-B}{vmax}$ time, since its velocity v in this controller is less than or equal to $vmax$. To prove this idea, I use a loop invariant to show that the evolution of the position of the dancer is always bounded to be less than the target in every iteration of the loop, and therefore it will not reach the target position in time. I chose the loop invariant $pos <= A + (t * vmax)$ because this is greatest position the dancer can reach, by picking the maximum velocity. Because $T < (B - A)/vmax$ and the system must stop at $t = T$, even if the dancer keeps moving at the maximum velocity, $T * vmax < (B - A)$. The displacement will not be enough to reach the target position.

Similarly, we prove using diamond notation that there exists some way for a dancer to reach the target position if the time interval is greater than $\frac{A-B}{vmax}$. The definitions, program variables, and controller remain the same in this model, but the value of T and the postcondition change. This model uses diamond notation because we only need to show that there exists some reachable state that satisfies the postcondition, not necessarily all reachable states.

Model 2 Robot in 1-D Can Reach in Time T

```
...
Problem
  (
     t=0 & v=0 & pos=A & T>0 & vmax>0 & A<B & vmax*T+A>=B
  ) /* Initial conditions */
  ->
  <
    {
      /* Controller */
      {v:=*; ?(v>=0 & v<=vmax);} /* Assign velocity */
      {pos'=v, t'=1 & t<=T & t>=0} /* Differential Equations */
    }*@invariant(t<=T & t>=0) /* Loop Invariant */
  > (pos>=B) /* Robot reaches end position */
End.
```

The intuition for this proof is also straightforward, as we can provide a value for v that allows the dancer robot to reach the postcondition. If the robot always chooses $vmax$, then if $T >= (B - A)/vmax$, the robot will always be at or exceed point B by time T. If the robot exceeds point B it must have reached point B at some point in its evolution, which also satisfies the condition that the robot can reach B within time T. I manually iterated through the loop and quantified the existence of values as I just described in order to prove this model. Note that the initial constraint on T is written slightly differently in the model to be easier to substitute for when proving in KeYmaera X.

4.2 2-D Case

4.2.1 Strategy for any Dancer to Reach Defined Point in 2-D

Now, I will model the movement of a single dancer in a 2-D space. We want to show that, given an arbitrary start and end point, it is possible for the dancer to travel from start to end in some way. I designed the controller to allow the dancer to change direction and velocity freely. Proving this model shows that it is a viable controller.

Model 3 Robot in 2-D Can Reach Defined Point

```
Definitions
  Real startx;  /* Start position of robot in x direction */
  Real starty;  /* Start position of robot in y direction */

  Real endx;  /* Desired end position of robot in x direction */
  Real endy;  /* Desired end position of robot in y direction */

  Real vmax; /* Maximum velocity in any direction */
End.

ProgramVariables
  Real rx; /* Position of robot in x direction */
  Real ry; /* Position of robot in y direction */

  Real vx; /* X velocity of robot */
  Real vy; /* Y velocity of robot */

End.

Problem
  (
     vmax>0 & rx=startx & ry=starty
  ) /* Initial conditions */
  ->
  <
    {
      {vx:=*; vy:=*; ?(abs(vx)<=vmax & abs(vy)<=vmax);} /* Assign velocity */
      {rx'=vx, ry'=vy} /* Differential Equations */
    }*>
  (rx=endx & ry=endy) /* Robot reaches end position */
End.
```

Because the model uses nondeterministic repetition, this also proves that, not only can the dancer reach the end point when it is at the start point, it can choose a path that brings it closer to the end point at any time the hybrid system is running, or at any position and velocity the dancer is at currently. While the intuition for this model and proof seems trivial, formally verifying the controller in diamond notation involved a substantial amount of reasoning about the position of the dancer from the end point. The main proof strategy uses convergence to show that there exists some distance constant n and some property $J(x)$ such that $J(n)$ holds initially, that $n <= 0$ implies the postcondition, and with every repetition, $J(n-1)$ holds. I defined $J(x)$ such that the distance from the dancer to the end point is less than or equal to n and lower bounded by 0, as scalar distance must be a positive value. So, if $n <= 0$, this means that the distance from the end point is 0, which implies the postcondition that the dancer has reached the desired end point. $J(n-1)$ holds after every subsequent repetition because the robot can pick a path in the direction of the end point, normalized to fit the velocity constraints, and evolve until it is less than or equal to $n-1$. I normalize the direction and velocity by making the velocity in the direction with the greatest distance from the target the magnitude of $vmax$, and I calculate the velocity in the other direction using the proportion of the slope of the line from the dancer to the target. I use cuts to divide the proof into cases where one direction is greater than another, as well as to case on whether each direction's velocity should be positive or negative. After dividing the proof into specific cases, I can calculate the normalized velocities and provide a witness for vx and vy that proves the existential formula and the convergence condition for $J(n-1)$.

I will not model and prove that a single dancer can reach a target point given a time constraint because, with just a single dancer and no obstacles, this reduces to the 1-D case. The linear path from start to end

is clearly the most efficient route. Proving a looser bound with forced path variation is not necessary as we would like this system to be as time-efficient as possible.

4.2.2 Multiple Dancers in 2-D

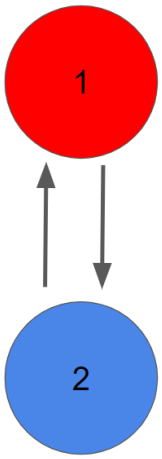

Figure 6: Two dancers swapping positions as modeled below.

Now, I will model that multiple dancers can travel to a destination in 2-D without colliding. Again, we define collision as point intersection. If the dancers are already at the target point, both their x and y velocity gets set to 0. Otherwise, each dancer always picks the linear path from its current position directly to the point, which by the previous proof, ensures that the dancer will reach the point. Then, the controller checks for path collision. Because the group is synchronized, we can reasonably assume that the robots will have some idea of where the other robots are going. In order to settle potential deadlock situations where robots will have to both stop and change direction, we establish a hierarchy such that the first robot travels freely, the second robot travels in a way that avoids collision with the first robot, the third robot avoids collision with the first and second robot, and so on. For all robots numbered $i \in [1, n]$, where n is the number of total robots, the ith robot avoids collision the first $i-1$ robots, and the next $n-i$ robots will avoid collision with the ith robot, therefore all possible combinations of robot collisions will be avoided in the overall system.

I will be modeling and proving a two-agent system. To make this situation interesting, I will be specifically modeling a situation where two dancers have to switch positions. So, initially, it is impossible for both of them to take the shortest path to their respective target positions. In the model setup, both dancer robots are on the same x axis, and dancer robot 1 is always above dancer robot 2 on the y axis. We can make this assumption because any pair of robots that switch positions can be transformed into this configuration, and we just assign the robot with the greater y position as robot 1.

Model 4 Two Robots in A Line in 2-D Space Do Not Collide

```
Definitions
  Real xaxis;  /* X position of both robots */
  Real starty1;  /* Start position of robot 1 in y direction, end y position of
      robot 2 */
  Real starty2;  /* Start position of robot 2 in y direction, end y position of
      robot 1 */
  Real vmax; /* Maximum velocity in any direction */
```

```
End.

ProgramVariables
  Real x1; /* Position of robot 1 in x direction */
  Real y1; /* Position of robot 1 in y direction */

  Real vx1; /* X velocity of robot 1 */
  Real vy1; /* Y velocity of robot 1 */

  Real x2; /* Position of robot 2 in x direction */
  Real y2; /* Position of robot 2 in y direction */

  Real vx2; /* X velocity of robot 2 */
  Real vy2; /* Y velocity of robot 2 */

End.

Problem
  (
     vmax>0 & x1=xaxis & y1=starty1 & x2=xaxis & y2=starty2 & starty1>starty2
  ) /* Initial conditions */
  ->
  [
    {
      /* Controller */

      /* Assign velocity 1. Always takes direct path to destination. */
      {vx1:=0;}
      {{?(y1=starty2);{vy1:=0;}} ++ /* If at end, stop */
      {?(y1!=starty2);{vy1:=-vmax;}}}

      /* Assign velocity 2. If already at point, stop. Take direct path to
         destination if possible.
      Find if robot intersects with other robot and when. */

      {vx2:=vmax*(xaxis-x2)/((xaxis-x2)^2+(starty1-y2)^2)^0.5;
      vy2:=vmax*(starty1-y2)/((xaxis-x2)^2+(starty1-y2)^2)^0.5;}
      {
          {?(x2=xaxis & y2=starty1); {vx2:=0; vy2:=0;}} /* If at end, stop */
            ++
        /* If paths collide choose another path */
        {?(!(x2=xaxis & y2=starty1));
        if ((vy2-vy1)*(xaxis-x2)=(vx2)*(y1-y2) & ((xaxis-x2)*(vx2)>0 | (y1-y2)*(
            vy2-vy1)>0)) /* If time of collision t>0 then must be same sign, if 0
             the paths will diverge or be the same speed*/

        {vx2:=*; vy2:=*; ?(abs(vx2)<=vmax & abs(vy2)<=vmax); ?((vy2-vy1)*(xaxis-
            x2)!=(vx2)*(y1-y2) | ((xaxis-x2)*(vx2)<0 & (y1-y2)*(vy2-vy1)<0));}}
      }

      {x1'=vx1, y1'=vy1, x2'=vx2, y2'=vy2} /* Differential Equations */
    }*@invariant(!(x1=x2 & y1=y2) & y1<=starty1 & x1=xaxis)
  ]
  (!(x1=x2 & y1=y2)) /* Robots do not collide */
End.
```

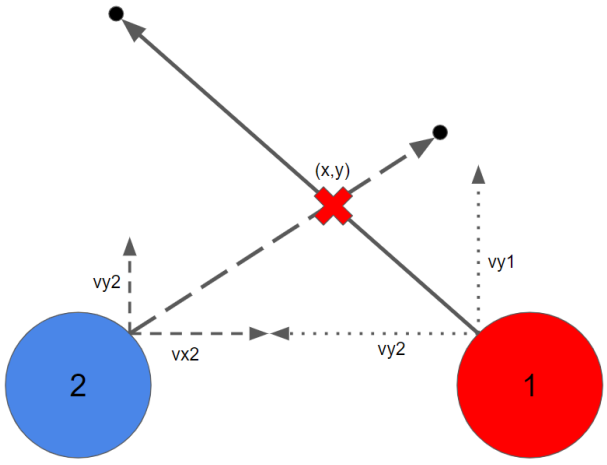

Figure 7: Intersection of two dancers' paths.

Figure 8: Visualization of x intersection of two dancers' paths.

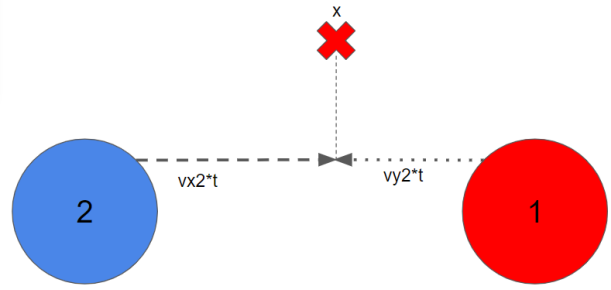

Figure 9: Visualization of y intersection of two dancers' paths.

This controller follows the hierarchy where Robot 1 always picks the path directly to its end point, so we can set its x velocity to 0 and let it have maximum y velocity. The controller for Robot 2 will pick the direct path to the end if possible, but if it intersects with Robot 1's path, it will arbitrarily choose a direction that does not intersect. I reasoned about path intersection using the time of intersection. Suppose two paths intersect. There must be a point of intersection (xi, yi). Then, there exists t such that $x1 + (vx1 * t) = xi$, and $x2 + (vx2 * t) = xi$. So $x1 + (vx1 * t) = x2 + (vx2 * t)$. From this, we can derive that $t = (x1 - x2)/(vx2 - vx1)$. Likewise, for the same t, $y1 + (yx1 * t) = yi$ and $y2 + (yx2 * t) = yi$, so $t = (y1-y2)/(vy2-vy1)$. t must be equal to itself so we get that $(x1-x2)/(vx2-vx1) = (y1-y2)/(vy2-vy1)$,

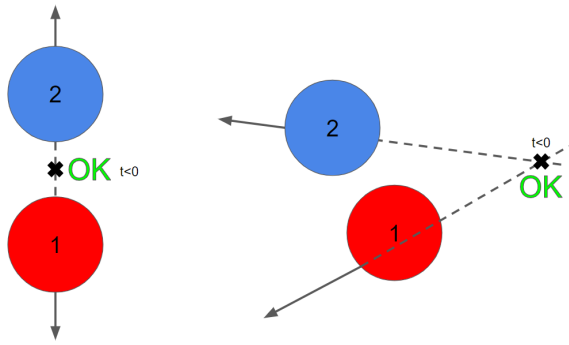

Figure 10: The two paths can intersect if the intersection time is less than 0 or in the opposite direction that the dancers are traveling.

or $(vy2 - vy1)(x1 - x2) = (vx2 - vx1)(y1 - y2)$, to remove division, which can be unsafe in KeYmaera X. If two paths intersect, their time of x intersection must be the same as their time of y intersection, so $(vy2 - vy1)(x1 - x2) = (vx2 - vx1)(y1 - y2) \implies$ intersection. However, even if this is true, Robot 1 and Robot 2 might not collide, because the time of collision could be negative. We have not yet figured out time travel, so we will only count a collision if it happens at $t > 0$. We can calculate t from the equations we reasoned about earlier. We find that $t = (x1 - x2)/(vx2 - vx1)$ and $t = (y1 - y2)/(vy2 - vy1)$. t represents the time displacement from the current state, so we also do not have to worry about any global time. We do not need to know the exact value of t, just that it is positive, so we can remove any unsafe division and just check that $(x1 - x2) * (vx2 - vx1) > 0$ and $(y1 - y2) * (vy2 - vy1) > 0$. Although t is the same for both x and y collision if the first condition is true, we need to check both because one could be zero if the paths are parallel. If both paths are 0 they either are colliding, which is not allowed from the initial conditions and loop invariant, or they are travelling in the same direction at the same vx and vy, so they will not intersect.

The proof for this model involves many branches based on the potential positions and velocities of the two robots. I first use the loop rule. The invariant for now says that the robots will not be at the same point and that Robot 1 will keep following the same path in one direction, but some more interesting conditions will replace this in parts of the proof. The loop invariant starts out general so that I can replace it using the monotonicity rule based on the conditions of both robots. There are several different general states the system can be in. Either Robot 1 is already at its endpoint, Robot 2 is already at its endpoint, both robots are at the end, or no robots are at the end. If both robots are at the end, they are both at rest and not at the same point, so trivially, they will not collide. If Robot 2 is already at its endpoint, it is at rest, and since Robot 2 handles all collisions, there is no way for it to collide with Robot 1, which continues in its direction until it reaches the end point. If Robot 1 is at its endpoint or if both robots are still moving, Robot 2 can either have a clear path to the end or intersect with Robot 1. If Robot 2 intersects with Robot 1, it must pick an arbitrary direction that does not intersect. I use differential invariants to show that Robot 2 is constantly moving away from Robot 1. If Robot 2 does not intersect with Robot 1, it stays with its original path. Then, for both these situations, if I did not already prove that Robot 2 is constantly moving away from Robot 1, I use the monotonicity rule to replace the postcondition with $(vy2 + vmax) * (xaxis - x2)! = vx2 * (y1 - y2)y1 <= starty1x1 = xaxis$, that after the loop the robots still do not collide. Throughout the proof, I cut information about the x and y positions of both robots so that KeYmaera can more easily reason about the intersection equation.

I originally planned to also model a system of 3 dancer robots, but it would follow generally the same proof strategy, with the third robot checking both the first and second robot, and this would require a very long repetitive proof, so I did not end up modeling it.

4.2.3 Multiple Dancers and Time Constraints

Now I will prove the time constraints for two dancers to swap positions. I will use the same controller as above but add time variables and change the initial condition, postcondition, and loop invariants. We already know that this controller is collision-safe so the postcondition of this model will only consider if the dancers can reach their positions in time.
This models that the dancers cannot switch positions in less than $T < (starty1 - starty2)/vmax$ time. We use box notation to show that, for any $t < T$, the dancers will not reach their end points.

Model 5 Two Robots in A Line in 2-D Space Do Not Collide

```
Definitions
  Real xaxis;  /* X position of both robots */
  Real starty1;  /* Start position of robot 1 in y direction, end y position of
     robot 2 */
  Real starty2;  /* Start position of robot 2 in y direction, end y position of
     robot 1 */
  Real vmax; /* Maximum velocity in any direction */
  Real T; /* Maximum time for transition */
End.

ProgramVariables
  Real x1; /* Position of robot 1 in x direction */
  Real y1; /* Position of robot 1 in y direction */

  Real vx1; /* X velocity of robot 1 */
  Real vy1; /* Y velocity of robot 1 */

  Real x2; /* Position of robot 2 in x direction */
  Real y2; /* Position of robot 2 in y direction */

  Real vx2; /* X velocity of robot 2 */
  Real vy2; /* Y velocity of robot 2 */

  Real t; /* Time */
End.

Problem
  (
     t=0 & T>0 & T<(starty1-starty2)/vmax & vmax>0 & x1=xaxis & y1=starty1 & x2=
        xaxis & y2=starty2 & starty1>starty2
  )
  ->
  [
    {
      /* Controller */
      {vx1:=0;}
      {{?(y1=starty2);{vy1:=0;}} ++ /* If at end, stop */
      {?(y1!=starty2);{vy1:=-vmax;}}}/
      {vx2:=vmax*(xaxis-x2)/((xaxis-x2)^2+(starty1-y2)^2)^0.5;
      vy2:=vmax*(starty1-y2)/((xaxis-x2)^2+(starty1-y2)^2)^0.5;}
      {
          {?(x2=xaxis & y2=starty1); {vx2:=0; vy2:=0;}}
            ++
        {?(!(x2=xaxis & y2=starty1));
        if ((vy2-vy1)*(xaxis-x2)=(vx2)*(y1-y2) & ((xaxis-x2)*(vx2)>0 | (y1-y2)*(
```

```
            vy2-vy1)>0))
          {vx2:=*; vy2:=*; ?(abs(vx2)<=vmax & abs(vy2)<=vmax); ?((vy2-vy1)*(xaxis-
              x2)!=(vx2)*(y1-y2) | ((xaxis-x2)*(vx2)<0 & (y1-y2)*(vy2-vy1)<0));}}
        }
        /* Differential Equations */
        {x1'=vx1, y1'=vy1, x2'=vx2, y2'=vy2, t'=1 & t<=T}
      }*@invariant(y1=starty1-t*vmax & x1=xaxis & y1>starty2 & y2<starty1 & y2<=
          starty2+t*vmax & t<=T)
    ]
    (!(x1=xaxis & y1=starty2 & x2=xaxis & y2=starty1)) /* Robots do not reach end
        position */
End.
```

In order to prove this, I take advantage of the fact that the robots are on the same x axis and must travel a linear distance in the y direction. The distance between the two positions is $(starty1 - starty2)$, and if the maximum velocity is $vmax$, it must take at least $(starty1 - starty2)/vmax$ time for a single robot to reach its destination. Therefore, if T is less than that, it is impossible for both robots to reach their destinations.

I also considered having $T <= (starty1 - starty2)/vmax$. When thinking about the proof for time constraints in 2D, I originally was reasoning about it as if the maximum velocity in any direction was $vmax$, which would mean that for any $T <= (starty1 - starty2)$, it would be impossible for both robots to swap places because Robot 2 would not be able to travel in a straight line or it would collide with Robot 1. If it travelled in a direction with $vxy > 0$, then $vxy < vmax$. Right from the beginning Robot 2 cannot pick the direction directly to its target position because it collides with Robot 1's path, so for any $t > 0$, $y2 < starty2 + vmax * t$. We would use this in the loop invariant to prove that y2 will never reach the target position within T, which means the two robots will not be able to swap positions in T time.

However, while proving this model, I realized that I assumed that $abs(vxy) > 0 \implies abs(vxy) < vmax$, when this was not actually true. In the controller, if the direct path to Robot 2's endpoint intersects Robot 1's path, Robot 2 can pick any vx and vy arbitrarily, with any magnitude less than or equal to $vmax$, as long as it does not collide with Robot 1. So, it is possible for Robot 2 to have nonzero vx as well as $vy = vmax$, so the invariant $y2 < starty2 + vmax * t$ would not work, and I replaced it with $y2 <= starty2 + vmax * t$ and $y2 < starty1$.

The main strategy used in this proof was, after using the loop rule, I would create branches from the different test conditions. The union split where the robots stop if they are at the target position already can be proved to be vacuously true using closeF as t would have to be T, and the y position of Robot 2 is strictly less than the y end position at T by the loop invariant. Then, with the remaining branches, I would cut in information about the positions of the robots, such as $y1 = starty1 - t * vmax$ $y2 <= starty2 + t * vmax$, which could be proved due to the unchanging motion of Robot 1 and that the maximum y velocity for Robot 2 is $vmax$, and then I would use differential weakening to show that these differential bounds implied the postcondition. This was also helpful in showing that the system remained within the time bounds. Using the differential invariant rule was often blocked by the fact that t was strictly increasing, and the constraint was that t be less than a value.

I also proposed a lower bound for allowing the robots to swap places successfully. Like in the 1-D case, I used diamond notation to show that there is at least one reachable state where the robots switch position. I proposed that the lower bound be $T > (starty1 - starty2)/vmax$. Earlier, I discussed that the model allows Robot 2 to pick $vy = vmax$ even if vx is nonzero. If Robot 2 can constantly pick $vy = vmax$, then this proof is fairly simple because Robot 2 can definitely reach the end point at $T = (starty1 - starty2)/vmax$ by picking $vy = vmax$. In 2 iterations of the repetition, Robot 2 can first pick some vx, and then pick $-vx$ to go back to $xaxis$.

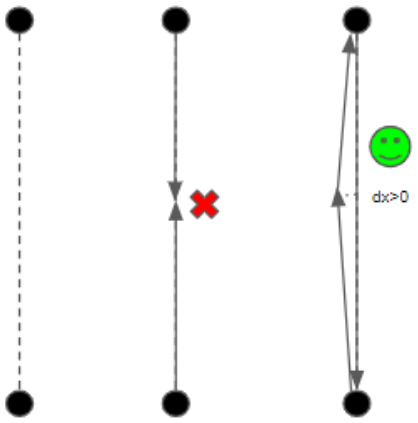

Figure 11: It will take Robot 2 more than $T = (starty1 - starty2)$ to reach the endpoint due to $dx > 0$

Although we can use the same unrolling strategy as in the 1-D case, this proof is not as simple as it initially appears, Earlier, I discussed that Robot 2 can always pick $vy = vmax$, but this is only when Robot 2 does not have a clear path to its endpoint. When Robot 2 has a clear path to the end point, the controller will pick a vector of magnitude $vmax$ in the direction of the target point instead of scaling the greater direction to $vmax$. Therefore, it requires $T > (starty1 - starty2)$ time to make sure both robots swap positions.

The exact time it takes depends on the x distance of the point from its target, as the greater the x distance, the smaller the y component of velocity. It is always possible to choose to travel a very small value of vx in the initial state, therefore reducing the additional time asymptotic to 0. My proof attempt was not able to prove the bound $T > (starty1 - starty2)/vmax$ but in my proof, I used $vmax$ for both vxy and vy initially, making vy on the return route $vmax/\sqrt{2}$ instead of $vmax$. This would require $t * \sqrt{2}$ more time than $vmax$, so the extra time would be $(t\sqrt{2} - t)$. I chose t such that $t = (starty1 - starty2)/(2 * vmax)$ to minimize the number of iterations in the proof to 3. In the first iteration, Robot 1 travels half the distance to the end point. Robot 2 also travels half its y distance because it can have $vy = vmax$. It also travels some x distance. In the second iteration, Robot 1 reaches its end point, and Robot 2 almost reaches its endpoint, based on the ratio between its x and y displacement. In the third iteration, Robot 1 stays in place and Robot 2 travels the rest of the distance to its endpoint, and we can pick a t that satisfies the intersection equations but with the end point. So, based on this, T in this proof would have to be $T >= (starty1 - starty2)/vmax + (\sqrt{2} - 1)(starty1 - starty2)/(2 * vmax) = (0.5 + \sqrt{2}/2)(starty1 - starty2)/vmax$.

4.3 Further Work

Here are some other extensions that I considered adding and that could be fully implemented in future work.

We could create a radial buffer area around each dancer to simulate the area a person takes up. This would essentially create two linear equations for each side of the person, which can be derived from the original equation of a point and the margin, the radius. We find the unit normal vector of the current line, and multiply that by the margin, and add it to both sides to find the starting points of these lines. Given the standard notation of a line $ax + by + c$ one line will be $ax + by + c0 = 0$ and one will be $ax + by + c1 = 0$. So the distance between these lines is $d = c0 - c1/\sqrt{a^2 + b^2}$. So we can derive that $r^2 = (c0 - c1)^2/\sqrt{a^2 + b^2}$, then $c1$ must be $c0 \pm r\sqrt{a^2 + b^2}$.

Although the ordering used in the proved model is arbitrary, we can also construct an ordering rule to improve the efficiency of the system without impacting safety. We establish an ordering such that the robots who are traveling backwards and robots with the furthest distance to travel are ranked higher in the hierarchy. In real life, dancers traveling backwards cannot react as quickly to obstacles behind them so we will try to allow them to account for as few obstacles as possible when choosing a path. Additionally, we want dancers with a long travel distance to be able to choose as close to linear a path as possible, so they will also account for fewer obstacles and minimize the time needed to complete the transition. The model for this is quite long and repetitive so we will restrict this to 4 dancers for now. Since this only affects the ordering we can use the same proof strategy.

Because of the number of similar agents in the system, another interesting area to explore would be QdL. Since we have already established an ordering system for the dancers, using QdL to quantify over the entire set of dancers and create constraints based on this ordering would allow greater model efficiency and less repetitive proofs. This could also model situations where a subset of dancers leave the center stage or if additional backup dancers join at some point in the dance.

5 Conclusion

I set out to and successfully modeled the motion of multi-agent transitions similar to those in K-Pop dance, and proposed reasonable time bounds for these transitions. This project provides a provable basis of work to potentially expand upon. There are still many interesting ideas to be explored that could not be formally proved within this project. It was certainly challenging to pivot to verifying diamond notation in contrast to the box proofs used in the rest of this course and it forced me to think of creative cuts, assignments, and convergences instead of relying on differential cuts and invariants based on the defined physical properties. With multiple robots, it was also challenging to think of helpful cuts that would separate the relationship between the two robots into cases that could be proved individually by differential invariant or differential weakening. I learned that there are many different approaches to a problem, and I tried several different ones before reaching the final version in this project. First I attempted a with a time-triggered linear 2-D controller that accounted for acceleration and buffer distance, and then I tried a circular-motion based controller similar to a modified version of Lab 5. However, I realized that verifying such approaches would not be easy or feasible in the time that I had to complete the project. Therefore, I successfully came up with a more conservative modeling approach that would allow me to verify basic motions and build onto them. The controllers that I proved could be extended to 2-D motion of robots in general, particularly time-constrained robots, perhaps emergency relief robots or robots handling time-sensitive tasks.

6 Deliverables

Note that proofs use unfold and auto to simplify syntax and prove that cuts are valid.

1d_lessT_proof.kyx - Model and proof that a robot cannot reach a defined point in 1D if the time constraint is less than some T.

1d_greaterT_proof.kyx - Model and proof that a robot can reach a defined point in 1D if the time constraint is larger than some T.

2dreachable_proof.kyx - Model and proof that a robot can reach a defined point in 2D, section 3.2.1.

2d_collisions_proof.kyx - Model and proof that two robots in 2D do not collide.

2d_lessT_proof.kyx - Model and proof that two robots in 2D cannot swap positions if the time constraint is less than some T.

2d_greaterT_partialproof.kyx - Model and **partial** proof that two robots in 2D can swap positions if the time constraint is greater than some T. Proof strategy may be hard to read, so please refer to section 3.2.3.

7 References

Platzer, Andre. (2012). A Complete Axiomatization of Quantified Differential Dynamic Logic for Distributed Hybrid Systems. Logical Methods in Computer Science, November 26, 2012, Volume 8, Issue 4. https://doi.org/10.2168/LMCS-8(4:17)2012.

Jorge CORTÉS, Magnus EGERSTEDT, Coordinated Control of Multi-Robot Systems: A Survey, SICE Journal of Control, Measurement, and System Integration, 2017, Volume 10, Issue 6, Pages 495-503, Released December 17, 2017, Online ISSN 1884-9970, Print ISSN 1882-4889, https://doi.org/10.9746/jcmsi.10.495

Yunus Emre Sahin, Petter Nilsson, and Necmiye Ozay. 2017. Provably-correct coordination of large collections of agents with counting temporal logic constraints. In Proceedings of the 8th International Conference on Cyber-Physical Systems (ICCPS '17). Association for Computing Machinery, New York, NY, USA, 249–258. DOI:https://doi-org.proxy.library.cmu.edu/10.1145/3055004.3055021