EXPERIMENTAL INVESTIGATIONS OF TRAJECTORY GUIDANCE AND CONTROL FOR DIFFERENTIAL GAMES

Neha Satak∗, Kurt Cavalieri∗, Clark Moody∗, Anshu Siddarth †, John E Hurtado ‡ and Rajnish Sharma †

The feedback solutions of Pursuit-Evasion games are of interest for formulating optimal strategies for the players in the game. In this paper feedback solution for a one pursuer and one evader zero-sum game is implemented on planar robots. The solution strategy exploits the differentially flat characteristic of robot models. The feedback strategy is demonstrated on hardware in real time using iRobot Create platforms. The effects of network latency and hardware limitations such as control bounds are studied. Results presented show the extent to which network latency and measurement noise affect the outcome of the game.

INTRODUCTION

Players in a game can have similar or different objectives. The game strategy chosen by a player not only depends on its objective but also on the objective of its opponents which may be known. In a zero-sum game1,2, the pursuer tries to minimize (or maximize) the same cost function that the evader tries to maximize (minimize). In this scenario, a saddle point, if it exists defines the best outcome any player can hope to achieve when the other player is also playing optimally. For the case when a player does not know its opponent’s goals, playing the zero-sum game may be pessimistic but ensures that the player at least can achieve the saddle point cost function. Therefore, the zero-sum game solution is important.

The work in this paper is a precursor to the efforts directed at developing differential game strategies suitable for hardware implementation. At outset of the development, a single pursuer and single evader zero-sum game is implemented in the laboratory using iRobot Create robotic platforms. The laboratory work demonstrates the advantages of feedback strategies as compared to open loop strategies in the presence of hardware limitations.

Linear quadratic (LQ) games are a special class of games for which each player’s dynamics is linear and their pay-off functions are quadratic. LQ games are comparatively

∗Graduate Research Assistant, Texas A&M University, Department of Aerospace Engineering, 3141 TAMU, College Station, TX, 77843-3141.
†Associate Professor, Texas A&M University, Department of Aerospace Engineering, 3141 TAMU, College Station, TX, 77843-3141, jehurtado@tamu.edu
‡Assistant Professor, Department of Aerospace Engineering and Mechanics, Hardaway Hall, Box 870280, The University of Alabama, Tuscaloosa, AL, 35487, rsharma@eng.ua.edu, AAS Member, AIAA Senior Member.
A feedback solution for a one pursuer and one evader LQ zero-sum game has been discussed in Bryson and Ho. For a differentially flat system there exist certain flat outputs that define all states and controls of the original system. Through the definition of the flat outputs, the nonlinear system of a differentially flat system can be modeled as a linear system. The planar robot kinematic equations are differentially flat and the flat outputs define a linear system model. The same linear model is used for solving the zero-sum game utilizing the planar robots in this paper.

The experiments are performed at the Land, Air, and Space Robotics (LASR) Laboratory at Texas A&M University. LASR provides the facilities necessary for hardware-based simulations of control algorithms. Experiments can be performed on several classes of robotic agents ranging from the simple, differentially driven iRobot Create to the more advanced Holonomic Omnidirectional Motion Emulation Robot. True position and orientation measurements required for proper implementations of feedback control can be obtained from three independent inertial sensing systems, each with distinct advantages. Nikon’s Indoor GPS (iGPS) system uses six rotating laser transmitters and four active sensors placed on the rigid body of interest. This system provides 0.5 mm inertial accuracy throughout the workspace at approximately 40 Hz. The PhaseSpace motion capture system uses eight cameras, each equipped with two linear imagers, along with active beacons that pulse at a unique frequency. PhaseSpace has the ability to compute measurements at a frequency of up to 480 Hz. The most versatile system, Vicon, uses six high-speed, 16 megapixel cameras that run at a 120 Hz update rate with random errors at the 1 mm level. Vicon rigid bodies are defined using passive, retro-reflective markers that are ideal for robotic systems that cannot support a heavy payload or provide a power source. The experimental area at the LASR Lab boasts a flat black, 2000 sqft workspace with 12 ft ceilings ideal for supporting any type of ground-based, space-bound hardware simulation.

This paper is divided into three sections. The first section explains the solution strategy used for the implementation. The second section discusses hardware limitations of the experimental platform used. The last section compares simulation results with those obtained by implementing in hardware.

P&E GAME

In this section, a one pursuer and one evader differential game is modeled and the control laws are formulated.

Model

The kinematic no-slip model of a planar robot is described by the following equations.

\[
\begin{align*}
\dot{x} &= u \cos \theta \\
\dot{y} &= u \sin \theta \\
\dot{\theta} &= \omega
\end{align*}
\]
where, $x$ and $y$ define the inertial position of the robot and $\theta$ denotes the orientation of the body reference frame with respect to the inertial reference frame. The forward velocity is denoted by $u$ and the turn rate by $\omega$. The differential flatness of this system allows the transformation into the following equivalent linear representation.

\[
\begin{align*}
\dot{x} &= \dot{X}_1 = X_3 \\
\dot{y} &= \dot{X}_2 = X_4 \\
\ddot{x} &= \dot{X}_3 = u_1 \\
\ddot{y} &= \dot{X}_4 = u_2
\end{align*}
\] (4-7)

Upon differentiating Eq. (1) and Eq. (2) and after some algebraic manipulations, the following relations can be obtained between the new controls $u_1$ and $u_2$, states $X_3$ and $X_4$ and the old controls $u$ and $\omega$.

\[
\begin{align*}
u &= \sqrt{X_3^2 + X_4^2} \\
\omega &= \frac{u_2 X_3 - u_1 X_4}{u^2}
\end{align*}
\] (8-9)

Since there is a division by $u^2$ in the expression for $\omega$, this transformation is valid only for $u \neq 0$. In order to satisfy this requirement the robots are prescribed with non-zero initial conditions both in hardware and in simulation.

**Game Definition**

The goal of the pursuer is to minimize its final distance from the evader whereas the evader has opposite intent. The following cost function $J$ captures their combined goals.

\[
J = \min_{u_p} \max_{u_e} [X_p(t_f) - X_e(t_f)]^T S_f [X_p(t_f) - X_e(t_f)] + \frac{1}{2} \int_0^{t_f} (u_p^T R_p u_p - u_e^T R_e u_e) dt
\] (10)

\[
S_f = W^T W
\] (11)

Here, $X = [X_1 \ X_2 \ X_3 \ X_4]$ defines the inertial position and velocity of the robot. The subscript “p” is for pursuer and “e” is for evader. The vectors $u_p$ and $u_e$ denote the pursuer’s and evader’s controls in the differentially flat domain.

\[
\begin{align*}
u_p &= [u_{1p} \ u_{2p}]^T \\
u_e &= [u_{1e} \ u_{2e}]^T
\end{align*}
\]
The matrix \( S_f \) is positive semi-definite and weighs the difference of pursuer’s and evader’s states at final time \( t_f \). The gain matrices \( R_p \) and \( R_e \) are positive definite weights on the pursuer’s and evader’s controls respectively. These three matrices define the game.

**Feedback Solution**

To obtain the feedback solution, the methodology described in Bryson and Ho\(^1\) is applied to the system model in this paper. Equations (4) to (7) can be written into the following linear system representation.

\[
\dot{X}_i = AX_i + Bu_i \quad i = p,e
\]  

\[
A = \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\] 

\[
B = \begin{bmatrix}
0 & 0 \\
0 & 0 \\
1 & 0 \\
0 & 1
\end{bmatrix}
\] 

Consider a new state vector \( z \) defined as\(^1\)

\[
\begin{align*}
\dot{X}_p(t) &= \Phi(t_f, t)X_p(t) \\
\dot{X}_e(t) &= \Phi(t_f, t)X_e(t) \\
z(t) &= W(\dot{X}_p(t) - \dot{X}_e(t))
\end{align*}
\]

where

\[
\Phi(t_f, t) = e^{A(t_f - t)}
\] 

Also, let

\[
\begin{align*}
P(t) &= W\Phi(t_f, t)B \\
E(t) &= W\Phi(t_f, t)B
\end{align*}
\]

From necessary conditions, the solution\(^1\) for a zero-sum game can be obtained by solving the following Riccati equation:

\[
\dot{S} = S(P(t)R_p^{-1}P^r(t) - E(t)R_e^{-1}E^r(t))S, \quad S(t_f) = I
\]
The controls for the pursuer and evader can then be calculated using Eq. (22) and Eq. (23) as

\[ u_p = -R_p^{-1}p^T\lambda \quad (22) \]
\[ u_e = -R_e^{-1}e^T\lambda \quad (23) \]

where, \( \lambda \) is the costate vector defined as

\[ \lambda = S(t)z(t) \quad (24) \]

**Example Game**

An example game is simulated with the following dimensional parameters:

\[ X_p(0) = \begin{bmatrix} -1.0200 & -1.7141 & -0.0853 & 0.0562 \end{bmatrix}^T \]
\[ X_e(0) = \begin{bmatrix} -0.4814 & -0.6704 & 0.0387 & 0.0927 \end{bmatrix}^T \]
\[ t_f = 10 \]
\[ R_p = \begin{bmatrix} 0.03 & 0 \\ 0 & 0.03 \end{bmatrix} \]
\[ R_e = \begin{bmatrix} 0.08 & 0 \\ 0 & 0.08 \end{bmatrix} \]
\[ W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \]

The gain matrices \( R_p \) and \( R_e \) are chosen such that the pursuer is faster than the evader. The resulting trajectories for the pursuer and evader are plotted in Figure 1 and the commanded control is shown in Figure 2. Intuitively, the evader should run directly away from the pursuer along the line connecting the pursuer and the evader. While the optimal action for the pursuer is to run behind the evader along the same direction with an orientation headed towards the evader. This behavior is demonstrated in Figure 1. A capture occurs at the final time and the final distance between the evader and pursuer is 0.0003 m. The value of this saddle point solution is the total cost function \( J \) that is 0.015. This is the ideal simulation cost for Case 1 in Section 3.

This simulation a priori assumes perfect measurements for both the players. There are no limits on the maximum control that can be commanded to the robots. Also, there is no latency in obtaining the measurements, control computation, or control implementation.
HARDWARE IMPLEMENTATION

In an effort to verify the validity of the theoretical solution, the game is implemented in hardware. Each player in the game is assigned an iRobot Create outfitted with an onboard netbook computer. The iRobot Create is an ideal platform for this particular application because of its differentially driven design. Control signals are computed on a centralized computer and are transmitted wirelessly to each robotic agent. Before performing any experiments, several challenges had to be addressed that distinguish the vast differences between computer- and hardware-based simulations.

Information flow

Since the iRobot Create is a lightweight, low power platform, the Vicon motion capture system was selected to provide position and orientation data. The inherit noise characteristics of the Vicon measurements and the lack of velocity information created the need of a filtering program to smooth the available state measurements and compute the rate of these states. Ideally, a filter which runs in parallel to the control algorithm is considered as the best choice so that the commanded forward and angular velocities could be delivered directly to the filter. However, because a Python\textsuperscript{5} framework was already in place to command the robots and a computationally quick filter would help cut down on network latency, a separate filtering routine was created and run independent of the control algorithm. The full control loop follows these steps:

1. Vicon senses the position and orientation of the robot and broadcasts the state data over the network.
2. The filter receives position and orientation data and updates the filtered state of each agent. It sends this state information to the control computer.

3. The control computer receives full state information from the filter and compute players controls, which it broadcasts over Wi-Fi to the robotic agents.

4. The computers on each robot receives forward velocity and turn rate commands then sends the proper serial command to the robot hardware.

All control and network communication software is written in the Python programming language except for the filter which is written in C.

Vicon and Filtering  Vicon provides full 6 degree-of-freedom (DOF) position and orientation information for each user defined rigid body in the experimental workspace. The orientation data for each agent is given in the form of a quaternion. For this particular application, the differential system moves in 3-DOF and each agent is outfitted with a custom platform to allow for a unique pattern of Vicon markers to define the rigid body. As mentioned earlier, there is inherent noise characteristics associated with the position and orientation information. Additionally, velocity data is required for the described control algorithm. These constraints make it necessary to implement a filtering routine to smooth the Vicon data and calculate the velocities necessary to compute the control commands to be sent to the robotic agents. An independent filter was written in C to help reduce computation time and ultimately, network latency. Because the filter has no knowledge of the actual velocity commands, a Gaussian least squares differential correction algorithm is used to estimate the commands $u$ and $\omega$ using a zero-jerk model and the 3-DOF Vicon measurements. From these command estimates, it is possible to solve for $x$, $y$, and $\theta$ and their associated rates analytically. These six states, for each agent, are then sent directly to the centralized control computer for command computation.

iRobot Create  On the other end of the game algorithm, the velocity commands need to be sent to the each player. As the central control computer produces motion commands, it broadcasts them over a wireless router to the robotic hardware agents. A control program on each agent computer translates these network packets into serial commands that are fed into the robot hardware. Since the serial connection to the hardware is rate-limited, the command frequency of the robot is capped at 20 Hz. The control program stores the most recently-received command and executes it once the control period has elapsed. This feature frees any off board controller from rate limitations; commands sent faster than the robot can respond to are simply discarded in favor of the most recent command.

Hardware Limitations

Control Mapping  The control program on each agent computer accepts velocity level commands, i.e., forward velocity $u$ and turn rate $\omega$ whereas the controls computed as a result of solving the game in the flat domain are acceleration commands in $x$ and $y$ direction, i.e., $u_1$ and $u_2$ respectively. The relation between these two sets of control variables comes
out as a result of the differential mapping (See Eq. (8) and Eq. (9)). This relation is again used below after substituting $\dot{x}$ and $\dot{y}$ in place of $X_3$ and $X_4$.

\[ u^2 = \dot{x}^2 + \dot{y}^2 \]  
\[ \omega = \dot{\theta} = \frac{u_2 \dot{x} - u_1 \dot{y}}{u^2} \]  

Differentiating $u^2$ provides

\[ u \ddot{u} = \dot{x} \ddot{x} + \dot{y} \ddot{y} \]  

Combining Eq. (26) and Eq. (27) into a matrix equation and rearranging it, the following relation is obtained:

\[ \begin{bmatrix} \ddot{u} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dddot{x} \\ \dddot{y} \end{bmatrix} = \begin{bmatrix} 1 \\ u \end{bmatrix} \begin{bmatrix} -\dot{y} & \dot{x} \\ \dot{x} & \dot{y} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \]  

The states $x$, $y$, $\dot{x}$ and $\dot{y}$ can be obtained through sensing and estimation. The current state information when used to compute $z(t)$ and $\lambda$, allows for the computation of controls $u_1$ and $u_2$ for each agent using Eq. (22) and Eq. (23). Therefore substituting Eq. (25) in Eq. (28), the following rates can be obtained:

\[ \dot{\theta} = \dot{\theta}(x, y, \dot{x}, \dot{y}, u_1, u_2) \]  
\[ \ddot{u} = \ddot{u}(x, y, \dot{x}, \dot{y}, u_1, u_2) = a \]  

Since the robots take $u$ and $\dot{\theta}$ as inputs, it is required to compute:

\[ \int_{t_1}^{t_2} a \ dt = \int_{u} \ddot{u} \ dt \]  
\[ u(t_2) - u(t_1) = a(t_2 - t_1) \]  

Here, $u(t_1)$ is the current measured forward velocity of the robot. For a small time step, a zero-order-hold is applied to calculate the commanded velocity. The velocity commanded at time $t_1$ is given by

\[ u = u(t_2) \]
Limited maximum velocity and turn rate  Hardware limitation on iRobot Create restricts the maximum forward velocity and maximum turn rate. These bounds are given as

\[ |u| \leq 0.5 \text{ m/s} \quad (34) \]
\[ |\omega| \leq 60 \text{ deg/sec} \quad (35) \]

Ideally, the solved optimal control problem should be subject to the maximum turn rate and maximum velocity hard constraints. But, in this implementation, the controls are subject to only soft constraints through the use of weights \( R_p \) and \( R_e \). If the computed control is higher than the maximum allowed then a saturation function clips the commanded to the closest maximum possible value. As a result, the vehicle will take longer to turn or traverse less distance than expected, but because the control is feedback in nature, the controller will be able to correct for it as it is computed based on the new updated position and orientation information. A future goal of the research is to perform a more rigorous development accounting for these constraints while solving the optimal control problem.

Therefore in the hardware implementation, the commanded forward velocity and turn rate are bounded. The commanded velocity, \( u_c \) and turn rate \( \omega_c \) are related to the computed controls \( u \) and \( \omega \) (from Eq. (33) and Eq. (9)) as shown below:

\[
u_{ci} = \begin{cases} 
0.5, & \text{if } u_i > 0.5 \\
u_i, & \text{if } |u_i| \leq 0.5 \\
-0.5, & \text{if } u_i < 0.5 
\end{cases}
\]

\[
\omega_{ci} = \begin{cases} 
60 \text{ deg/sec}, & \text{if } \omega_i > 60\text{deg/sec} \\
\omega_i, & \text{if } |\omega_i| \leq 60\text{deg/sec} \\
-60\text{deg/sec}, & \text{if } \omega_i < 60\text{deg/sec} 
\end{cases}
\]

where \( i =p,e \).

Network Latency  Network latency plays a significant role in testing the robustness of any control scheme. By the time the robot is commanded, the command computation is already an old command which was computed with even older measurements. Even though we cannot completely avoid the communication framework of latency, extra care can be taken to reduce latency in the hardware simulation. The C-based filter routine provides a faster computation time than one written in the Python language. Also, data is sent directly to the centralized control computer and each computer associated with the different players using a UDP protocol in favor of a TCP connection. Furthermore, latency can be added to the simulation to aid in the comparison of computer- and hardware-based results. The latency associated with the entire framework has been characterized to be approximately 0.20 sec.

9
HARDWARE SIMULATION RESULTS

Two example hardware cases are analyzed and compared with their computer-based simulation using the identical initial conditions. To make the simulation realistic, the commanded controls are computed using Eq. (36) and Eq. (37) from the computed controls. Additionally, the network latency is incorporated in the simulation. The measurements are delayed for the computation of controls by 0.1 sec, hence depicting the network latency in obtaining Vicon’s filtered measurements. The computed controls are also delayed for application by 0.1 sec, in order to depict the network latency in the robots receiving the commands.

Case 1

The initial conditions for this example case are equivalent to those used for the ideal simulation example shown in the first section. These initial conditions are:

\[
X_p(0) = \begin{bmatrix} -1.0200 & -1.7141 & -0.0853 & 0.0562 \end{bmatrix}^T
\]

\[
X_e(0) = \begin{bmatrix} -0.4814 & -0.6704 & 0.0387 & 0.0927 \end{bmatrix}^T
\]

\[
t_f = 10
\]

\[
R_p = \begin{bmatrix} 0.03 & 0 \\ 0 & 0.03 \end{bmatrix}
\]

\[
R_e = \begin{bmatrix} 0.08 & 0 \\ 0 & 0.08 \end{bmatrix}
\]

\[
W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}
\]

The actual trajectories taken by the iRobot Create are compared with the realistic simulation results in Figure 3. The pursuer as well as the evader, in the experiment and in simulation, traverses the same general direction. The initial position and orientation \( \theta \) (See Figure 4), of the pursuer (\( \theta_p(0) = 146.6 \) deg.) and evader (\( \theta_e(0) = 67.4 \) deg.) suggest that the pursuer should turn in order to follow the evader while the evader should just run forward. This is confirmed in the commanded control histories in Figure 5 and the trajectories of the robots in Figure 3. The distance between the pursuer and the evader at final time is 0.85 m in experiment and 0.52 m in simulation. This difference in the final distance is attributed to the difference in the commanded velocity profile shown in Figure 5. While the simulation assumes perfect but only delayed measurements, in actual experiment, there is also noise in the filtered velocity data due to the noisy position measurements. This noise is difficult to capture in simulation due to its randomness. Since a feedback control is applied
to the system, a noisy velocity measurement affects the complete trajectory. Figure 6 shows
the commanded forward velocity and turn rate overlaid with the measured forward velocity
and turn rate. Except the measurement noise the trend followed by the commanded and
implemented control is similar. As can be seen from Figure 3 to 5, the simulated and actual
game results follow the same trend.

The value of the total cost function $J$, defined in Eq. (11), for hardware experiment is
0.3693. Recall that this value was 0.015 for the ideal simulation in the first section. This
shows that the actual game in hardware gets tilted towards the evader, who is trying to
maximize $J$. This is also apparent from the final distance of 0.85 m as compared to near
0 distance in ideal simulation. These are the effects of control saturation, network latency
and measurement noise. The total cost function value for realistic simulation results plotted
in this section is 0.1469. This is mid-way between the ideal simulation and the actual
hardware. This is correct as the realistic simulation includes network delay, but it still does
not include the measurement noise experienced in the actual hardware experiment.

<table>
<thead>
<tr>
<th>Raw Text</th>
<th>Natural Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 3: Case 1: $x$-$y$ trajectory</td>
<td>Figure 4: Case 1: Simulated and measured states</td>
</tr>
</tbody>
</table>

**Case 2**

In the second game, the pursuer and evader will both need to turn in order to chase and
get away from each other respectively. This is because, at the start of the game, they both
stand facing in the direction perpendicular to the line joining them. The initial conditions
Figure 5: Case 1: Commanded control histories in simulation and in experiment

for this game are given below.

\[ \mathbf{X}_p(0) = \begin{bmatrix} -0.7878 & -1.6323 & 0.0628 & 0.0705 \end{bmatrix}^T \]

\[ \mathbf{X}_e(0) = \begin{bmatrix} 0.2806 & -0.9175 & -0.0761 & 0.0625 \end{bmatrix}^T \]

\[ t_f = 10 \]

\[ R_p = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix} \]

\[ R_e = \begin{bmatrix} 0.08 & 0 \\ 0 & 0.08 \end{bmatrix} \]

\[ W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \]

In this example, due to the choice of \( R_p \) and \( R_e \), the pursuer is still faster than the evader but only marginally. As a result, though the pursuer manages to reduce the distance from the evader by 1.285 m (at the start of game) to 0.95 m (at the end) but this decrement is not substantial. Figure 7 shows that the simulated results match closely to the actual hardware results. The final distance between the pursuer and the evader in simulation is 0.71 m while in experiment is 0.95 m. In Figure 9, the commanded forward velocity of pursuer and evader monotonically increases until the maximum limit is reached at around 7 seconds for pursuer and around 8 seconds for the evader. Looking at the history of commanded turn rate \( \omega \) in Figure 9, it is found that the robots orient themselves while moving forward
slowly in the first 3 seconds. After this, they move forward without any turn rate. This behavior matches with the simulated commands overlaid in the same figure. The value of the cost function $J$ for hardware experiment is 0.4373 whereas for the realistic simulation it is 0.2433. This difference occurs due to measurement noise affecting the optimality of the solution. It is found that the ideal simulation incurs a total cost of 0.0092.

CONCLUSIONS AND FUTURE WORK

The one pursuer and one evader game was implemented in the LASR laboratory using iRobot Create platforms as agents. Hardware limitations encountered during this implementation emphasize the need for developing feedback game strategies taking realistic control bounds and measurement noise into account. It is found that the robot trajectories, commanded velocities and turn rates in hardware results match more closely with simulation when the network latency is included in the simulation.

Future work will involve incorporating bounds on controls and states in the formulated differential game. Further work will also employ techniques to combat latency issues.

REFERENCES

Figure 7: Case 2: $x$-$y$ trajectory

Figure 8: Case 2: Simulated and measured states

Figure 9: Case 2: Commanded control histories in simulation and in experiment

Figure 10: Case 2: Commanded control in experiment compared with measured rates