Corralling Sheep with Optimal Control

(in Two and Three Dimensions)

Danny Perkins, Trevor Garrity, Drake Brown, Davis Hunter, Wyatt Pochman

Overview

Many a shepherd has asked themself the question, "what are the optimal routes for my m dogs to take to corral my n indignant sheep into their pen?"

We first work in a 2D environment with no obstacles or boundaries.

The Problem

Modeling Assumptions

We begin by considering the behavior of the sheep. We assume each sheep wishes to maximize their distance from the dogs at each time t, where they weigh the closest dog(s) higher (see Figure 3). Because sheep commonly travel in groups, we enforce no constraint on the distances between sheep.

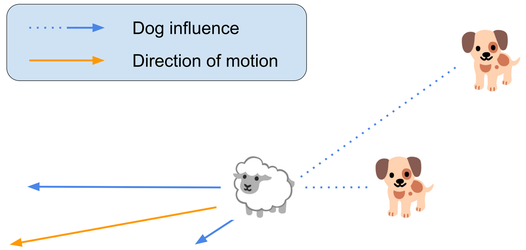

Figure 1: A visualization of the sheep strategy for $m = 2$ and $n = 1$.

Let sheep i's position be given by the two-dimensional vector $\mathbf{s}^{(i)} = (s_x^{(i)}, s_y^{(i)})$, and let dog j's position be given by the two-dimensional vector $\mathbf{d}^{(j)} = (d_x^{(j)}, d_y^{(j)})$. We model the sheep strategy from Figure 3 using the following second-order ODE giving the acceleration of sheep i:

$$(\mathbf{s}^{(i)})'' = \sum_{j=1}^{m} \frac{\mathbf{s}^{(i)} - \mathbf{d}^{(j)}}{\|\mathbf{s}^{(i)} - \mathbf{d}^{(j)}\|^{\lambda} + \epsilon} \tag{1}$$

We fix the degree term $\lambda \in \mathbb{Z}$ and the smoothing parameter $\epsilon > 0$ to avoid division by zero. The higher the value of λ is, the more attentive the sheep are to the nearest dog (and less attentive to the rest of the dogs). In practice, we find that $\lambda = 3$ and $\epsilon = 10^{-3}$ give a good balance. In this setting, the above acceleration approximates that of a charged particle in the presence of other like-charged particles.

The control problem is to find the optimal acceleration $\mathbf{u}^{(j)} = (\mathbf{d}^{(j)})''$ of each dog j, given the following goals:

- The dogs wish to guide the sheep towards the origin (sheep pen);
- The dogs wish to stay relatively close to the origin;
- The dogs wish to limit their acceleration to preserve energy.

One potential corralling method is given in Figure 2.

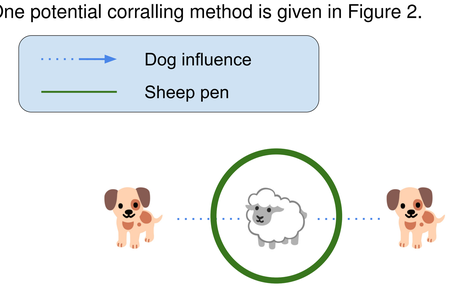

Figure 2: One potential dog strategy for $m = 2$ and $n = 1$.

To accomplish our goals, we wish to solve the following problem:

$$\text{minimize} \quad \int_0^{t_f} \left(\alpha \sum_{i=1}^{n} \|\mathbf{s}^{(i)}\|_2^2 + \beta \sum_{j=1}^{m} \|\mathbf{d}^{(j)}\|_2^2 + \sum_{j=1}^{m} \|\mathbf{u}^{(j)}\|_2^2 \right) dt \tag{2}$$

$$\text{subject to} \quad \begin{pmatrix} d_x^{(j)} \\ d_y^{(j)} \\ (d_x^{(j)})' \\ (d_y^{(j)})' \end{pmatrix}' = \begin{pmatrix} (d_x^{(j)})' \\ (d_y^{(j)})' \\ u_x^{(j)} \\ u_y^{(j)} \end{pmatrix} \quad \forall 1 \le j \le m, \tag{3}$$

$$\begin{pmatrix} d_x^{(j)} \\ d_y^{(j)} \end{pmatrix} = \mathbf{d}_0^{(j)} \quad \forall 1 \le j \le m, \tag{4}$$

$$\begin{pmatrix} s_x^{(i)} \\ s_y^{(i)} \\ (s_x^{(i)})' \\ (s_y^{(i)})' \end{pmatrix}' = \begin{pmatrix} (s_x^{(i)})' \\ (s_y^{(i)})' \\ \sum_{j=1}^{m} \frac{s_x^{(i)} - d_x^{(j)}}{\|\mathbf{s}^{(i)} - \mathbf{d}^{(j)}\|^3 + 10^{-3}} \\ \sum_{j=1}^{m} \frac{s_y^{(i)} - d_y^{(j)}}{\|\mathbf{s}^{(i)} - \mathbf{d}^{(j)}\|^3 + 10^{-3}} \end{pmatrix} \quad \forall 1 \le i \le n, \tag{5}$$

$$\begin{pmatrix} s_x^{(i)} \\ s_y^{(i)} \end{pmatrix} = \mathbf{s}_0^{(i)} \quad \forall 1 \le i \le n, \tag{6}$$

for some fixed time t_f. Note that equation (5) is equivalent to equation (1) expressed as a first-order ODE; however, the state space is now $4(m+n)$-dimensional and the co-state space is also $4(m+n)$-dimensional. Here we have prescribed the controls

$$u_x^{(j)} \equiv \text{acceleration of dog } j \text{ in the } x\text{-direction}, \quad \forall 1 \le j \le m,$$
$$u_y^{(j)} \equiv \text{acceleration of dog } j \text{ in the } y\text{-direction}, \quad \forall 1 \le j \le m.$$

The parameters α and β are free (with $\alpha \ge \beta$); fitting values are determined through experimentation. Using the setup from Pontryagin's Maximum Principle, we attempt two strategies for solving the problem.

Solving Numerically with `solve_bvp`

We begin by using the SciPy function `scipy.integrate.solve_bvp` to numerically solve the original, nonlinear problem. This function is based on a collocation algorithm [KS01]. Essentially, the solution is approximated with a piecewise cubic polynomial, whereupon the differential equation evaluated on a set of mesh points becomes an algebraic equation that can be solved exactly. Although versatile, the algorithm may require a large number of mesh points, which becomes expensive in terms of both computation and memory.

Using SymPy, we analytically compute and simplify the partial derivative of the Hamiltonian with respect to the control variable $\frac{DH}{Du}$. We then set $\frac{DH}{Du} = 0$ to find the optimal control $\tilde{u}$ in terms of the state **x** and co-state **p**. Finally, we define the state equations (3) and (5) and their corresponding co-state equations from Pontraygin's Maximum Principle, which we solve to to determine each $\mathbf{s}^{(i)}$ and $\mathbf{d}^{(j)}$.

Solving Numerically with Linearization and Infinite-Time LQR

Notice that the cost functional (2) is quadratic in all variables and the dog state evolution equation (3) is linear. The sheep equation (5) is nonlinear, but we can linearize it about each point, allowing us to apply LQR to approximate the optimal control at each time step.

To make use of LQR, it's also important to extend the cost functional over infinite time horizon. The reason is that solving the finite-horizon case would require solving an initial value problem (IVP) for the P matrix. Since $A(t) = Df(x(t))$ is state-dependent at each time step, we'd need the full trajectory in advance to compute the control. To get around this, we consider infinite-horizon LQR, where we only need to solve the Algebraic Ricatti Equation at each time step. (Alternativelty, we could define some initial trajectory and use an iterative algorithm to update the controls accordingly.)

Early Results & Analysis

`solve_bvp`

The preliminary results for the `solve_bvp` method are encouraging. Starting from a random initial guess, the resulting solution has two dogs taking intelligent paths to immediately begin herding the sheep toward the origin. Although the visualization appears promising, the solver does not fully converge. This is due to `solve_bvp` stopping as soon as the maximum number of mesh nodes is reached, leading the algorithm to return a "solution" whose derivative poorly approximates the differential equation. Additionally, performance degrades when the numbers of dogs or sheep is increased.

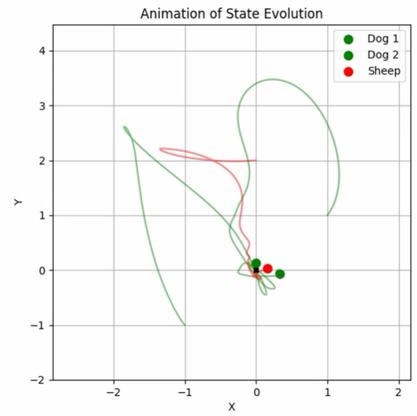

Figure 3: A visualization of the output from the `solve_bvp` method with $m = 2$ dogs and $n = 1$ sheep. The two dogs position themselves in front of the sheep and progressively steer it toward the origin, circling around it as it nears the destination.

Linearization and Infinite-Time LQR

After implementing the linearized LQR solver, we are able to add more sheep and dogs to the simulation, but convergence is very slow. We find linearization of equation (5) introduces difficulties when the dogs were very close to the sheep.

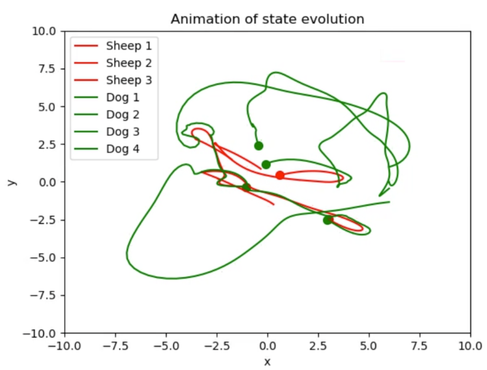

Figure 4: A visualization of the herding strategy with $m = 4$ dogs and $n = 3$ sheep. We find that typically one or two dogs will go after each sheep and guide them back to the origin.

Extension to Three Dimensions

We can naturally extend the simulation to 3D by augmenting the state equations to include a z-dimension term in equations (3) and (5).

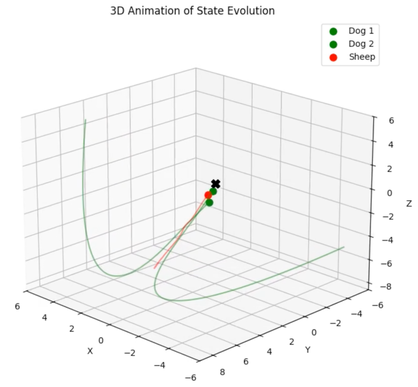

Figure 5: A visualization of the output from the `solve_bvp` method in 3 dimensional space. The two dogs position themselves in behind the sheep and push it toward the origin.

Conclusions and Next Steps

Using Pontryagin's Maximum Principle, we can solve for a differential equation determining the optimal equations of motion for a group of dogs herding a group of sheep. We numerically approximated the solution in the case of two dogs and one sheep using two different methods: a collocation method used by SciPy's `solve_bvp` and an iterative LQR method (with an infinite time horizon). For some initial conditions, the solutions displayed the desired long-term behavior but failed to be locally optimal, likely due to algorithmic convergence issues resulting in poor approximations to the differential equation.

There are a couple improvements we can make. First, we can more carefully use `solve_bvp` by providing better initial guesses, increasing the number of mesh nodes, and iteratively applying the algorithm with improved guesses. We can also use JAX to supply the exact Jacobian matrix, improving both speed and accuracy. Second, we can explore other numerical methods, such as the direct shooting method [BP84], which divides the time interval into sub-intervals and solves initial value problems using Newton's method. Finally, we could change the cost functional itself by adjusting hyperparameters or replacing integral costs with endpoint costs.

References

[BP84] Hans Georg Bock and Karl-Josef Plitt, *A multiple shooting algorithm for direct solution of optimal control problems*, IFAC Proceedings Volumes **17** (1984), no. 2, 1603–1608.

[KS01] Jacek Kierzenka and Lawrence F Shampine, *A bvp solver based on residual control and the maltab pse*, ACM Transactions on Mathematical Software (TOMS) **27** (2001), no. 3, 299–316.