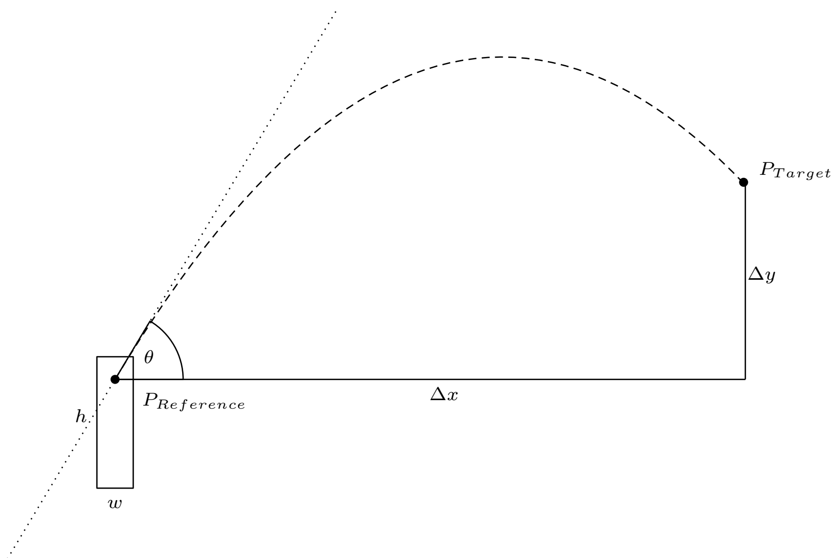
1 Trajectory Planner

The result of $TrajectoryPlanner.estimateLaunchPoint()$ is used to construct $Shot$s

$$Shot(P_{Reference}, P_{Release} - P_{Reference}, t_1, t_2), \tag{1}$$

where t_1 is the time when the shot is performed and t_2 is the time when the special ability of the bird is triggered, that are given to the $SimulationManager$ to simulate them and estimate their score.

1.1 Converting The Trajectory Into Our Simulation



The resulting parabola of the trajectory planner can be seen in the function $TrajectoryPlanner.setTrajectory()$

```
public void setTrajectory(Rectangle sling, Point
    releasePoint) {
     if (_trajSet && _ref != null &&
         _ref.equals(getReferencePoint(sling)) &&
         _release != null
              && _release.equals(releasePoint))
          return;
     _scale = sling.height + sling.width;
     _ref = getReferencePoint(sling);
     _release = new Point(releasePoint.x,
         releasePoint.y);
```

```
        _theta = Math.atan2(_release.y - _ref.y,
            _ref.x - _release.x);
        _theta = launchToActual(_theta);
        _velocity = getVelocity(_theta);
        _ux = _velocity * Math.cos(_theta);
        _uy = _velocity * Math.sin(_theta);
        _a = -0.5 / (_ux * _ux);
        _b = _uy / _ux;
        _trajectory = new ArrayList<Point>();
        for (int x = 0; x < X_MAX; x++) {
            double xn = x / _scale;
            int y = _ref.y - (int) ((_a * xn * xn +
                _b * xn) * _scale);
            _trajectory.add(new Point(x + _ref.x, y));
        }
        _trajSet = true;
    }
```

In short it is given by the equation:

$$y_{px}(x) = \frac{1}{2 * u_x^2 * (h+w)} * x_{px}^2 - \frac{u_y}{u_x} * x_{px}, \tag{2}$$

where $\vec{\boldsymbol{u}}$ is the velocity in the koordinate system of the trajectory planner. The units of the simulation are meters and not pixels thus the parabola needs to be converted with $y_m = \frac{y_{px}}{ppm}$ and $x_m = \frac{x_{px}}{ppm}$, where ppm are the pixels per meter, and since the y-axis of the vision is upside down we need the negetive value of y:

$$y_m(x) = \frac{-y_{px}(x)}{ppm} \tag{3}$$

$$y_m(x) = \frac{-\frac{1}{2*u_x^2*(h+w)} * x_{px}^2 + \frac{u_y}{u_x} * x_{px}}{ppm} \tag{4}$$

$$y_m(x) = -\frac{1}{2 * u_x^2 * (h+w)} * \frac{x_{px}^2}{ppm} + \frac{u_y}{u_x} * \frac{x_{px}}{ppm} \tag{5}$$

$$y_m(x) = -\frac{ppm}{2 * u_x^2 * (h+w)} * x_m^2 + \frac{u_y}{u_x} * x_m \tag{6}$$

Any shot in the simulation can be expressed by the following equation:

$$y_m(x) = -\frac{g}{2 * v_x^2} * x_m^2 + \frac{v_y}{v_x} * x_m. \tag{7}$$

To perform the *Shot* given to the simulation the parameters g and v need to be calculated. From our earlier measurements we have concluded, that $g = 9.81\frac{m}{s^2}$ given a slingshot height of $5m$. This leaves only v to be calculated. From the equations (6) and (7) follows that

$$-\frac{g}{2 * v_x^2} = -\frac{ppm}{2 * u_x^2 * (h+w)} \tag{8}$$

$$\frac{v_y}{v_x} = \frac{u_y}{u_x} \tag{9}$$

which can be solved for v

$$v_x^2 = \frac{g * (h+w)}{ppm} * u_x^2 \tag{10}$$

$$\Updownarrow \text{ since } v \text{ and } u \text{ show in the same direction} \tag{11}$$

$$v_x = \sqrt{\frac{g * (h+w)}{ppm}} * u_x \tag{12}$$

$$v_y = \frac{u_y}{u_x} * v_x \tag{13}$$

$$v_y = \sqrt{\frac{g * (h+w)}{ppm}} * u_y \tag{14}$$

$$\vec{\boldsymbol{v}} = \sqrt{\frac{g * (h+w)}{ppm}} * \vec{\boldsymbol{u}} \tag{15}$$