Learning Contact Dynamics with LCP Constraint Relaxations

Samuel Pfrommer, Michael Posa

July 23, 2019

DAIR Lab at the University of Pennsylvania

Overview

- Problems with manipulation
- Existing approaches
- How are we learning?
- What are we learning?

Manipulation overview

Problems with manipulation

- Sudden changes in dynamics when making/breaking contact
- Inconsistencies with Coulomb friction (Painlevé paradox)
- Many simultaneous contacts
- Stick/slip transitions

Existing approaches

Learned

- Often in context of policy learning
- Slow and data innefficient
- Doesn't leverage existing understanding of contact dynamics

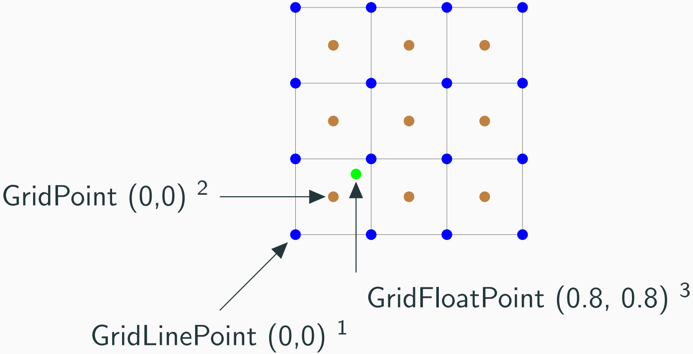
Hybrid

- Best of both worlds
- Residual physics
- Sim-to-real
- **Differentiation through LCPs**

Analytical

- Only an approximation
- Doesn't fully capture real-world phenomena

GridPoint, GridLinePoint, GridFloatPoint



GridPoint (0,0) [2]

GridFloatPoint (0.8, 0.8) [3]

GridLinePoint (0,0) [1]

[1] A new type

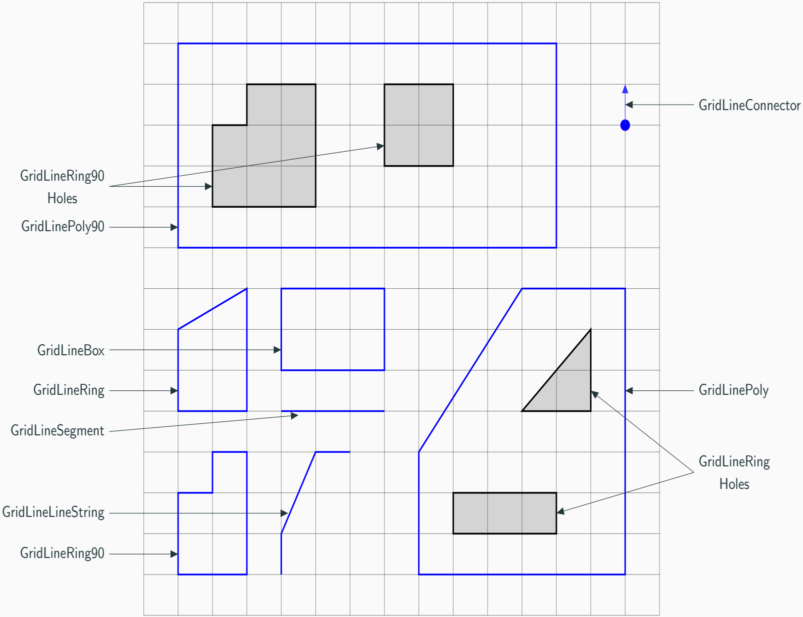
[2] Reuse of existing occupancy-grid GridPoint

[3] Reuse of existing occupancy-grid grid_coord_t

Derived geometries

- All associated geometries have names derived from point type and characteristics
    - GridPointSegment vs GridLineSegment
    - GridPointRing/GridPointRing90 vs GridLineRing/GridLineRing90
    - GridPointPoly/GridPointPoly90 vs GridLinePoly/GridLinePoly90
    - etc...

Gridline geometries examples



GridPoint and GridLinePoint

What is difference between calling boost::geometry::within on a GridPoint and GridLineRing and calling boost::geometry::within on a GridLinePoint and GridLineRing?

- The GridPoint (0,0) is spatially shifted by 0.5f in x and y from the GridLinePoint (0,0)
- Before using boost geometry functions relating GridPoints and GridLinePoints, GridPoints must be brought into the GridLinePoint reference frame (see type_conversions.h)
- room-segmentation/geometry/algorithms/within and room-segmentation/geometry/algorithms/covered_by handle this automatically

Custom base geometry types

- Polygon
    - Has outer ring and vector of inner rings
    - Had to be custom since boost geometry polygon hard coded ring types internally (and we really only want a GridLinePoly90 to take GridLineRing90s)
- Ring and Linestring
    - Implemented using shared_ptr to point vector rather than inheriting from vector
    - Allows for shallow copies and conversion between Ring and LineString representation without deep copying points (explain why necessary later)

Sample API

moment.h

```
float moment_x(const GridLineRing& ring);
float moment_x(const GridLineRing90& ring);
```

moment.cpp

```
template <typename RingT> float moment_x_generic(const RingT& ring)
{
    float sum = 0.0f;
    for (size_t i = 0; i < ring.size() - 1; i++)
    {
        float x_i, y_i, x_in, y_in = ...
        sum += (y_i*y_i + y_i*y_in + y_in*y_in) * (x_i*y_in - x_in*y_i);
    }
    sum = sum / 12.0f;
    return -sum;
}
float moment_x(const GridLineRing& ring)
{
    return moment_x_generic(ring);
}
float moment_x(const GridLineRing90& ring)
{
    return moment_x_generic(ring);
}
```

Shallow copy usage scenarios

- Do **not** need type reinterpreting for generic APIs operating on geometries
- Might need type reinterpreting to set an existing geometry variable
- Use sparingly: really should only convert rectilinear to nonrectilinear

regionWshed.h

```
class RegionWshed
{
    void set_simplified_boundary(const GridLinePoly& p);
    void set_simplified_boundary(const GridLinePoly90& p);

    GridLinePoly _simplified_boundary;
}
```

regionWshed.cpp

```
void RegionWshed::set_simplified_boundary(const navgeometry::GridLinePoly90& p)
{
    _simplified_boundary = reinterpret_poly<GridLinePoly>(p);
}
void RegionWshed::set_simplified_boundary(const navgeometry::GridLinePoly& p)
{
    _simplified_boundary = p;
}
```

Reinterpreting geometries

- LineString and Ring are both just vectors of points
- Boost geometry algorithms sometimes behave differently and one behavior might be preferred
- Use helper function reinterpret_geometry

```cpp
GridLineRing90 ring({{0, 0}, {0, 4}, {4, 4}, {4, 0}, {0, 0}});
GridLineLineString ring_reinterpreted = reinterpret_ring<GridLineLineString>(ring);
// Outputs zero since point is inside ring
cout << boost::geometry::distance(GridLinePoint(1, 1), ring1) << endl;
// Outputs one (desired)
cout << boost::geometry::distance(GridLinePoint(1, 1), ring1_reinterpreted) << endl;
```

API structure

Interacting with grids

Traversal directions

```
enum class GridDir { RIGHT = 0, UP = 1, LEFT = 2, DOWN = 3 };
enum class DiagDir { UPRIGHT = 0, UPLEFT = 1, DOWNLEFT = 2, DOWNRIGHT = 3 };
enum class TurnDir { STRAIGHT = 0, LEFT = 1, REVERSE = 2, RIGHT = 3 };
```

grid_traversal.h

- Various utility functions for traversing grids

type_conversions.h

- Converting GridPoints to GridFloatPoints
- Also for shallow converting different types of geometries

Boost geometry wrappers

- Calling boost::geometry::within(GridPoint p, GridLineRing r) is **WRONG**
- Say the GridPoint is (2,0), and a segment of the GridLineRing goes from (0,0) to (4,0)
- Boost geometry will say that the point is not within the polygon since it's on the boundary, even though the GridPoint should be shifted up and to the right by 0.5 and hence be within
- Same applies to boost::geometry::covered_by and other similar functions
- Solution: use custom navgeometry::within or navgeometry::covered_by (in geometry/algorithms)

Adding algorithms

- All algorithms go in geometry/algorithms
- Try to use verbs for the file names wherever possible
- **ONE** algorithm, **ONE** .h/.cpp file pair
- .h file declares what geometry types the algorithm can take
- If implementations for rectilinear/nonrectilinear geometries are different, have two separate implementations in .cpp file; otherwise, have a templatized version **IN THE CPP FILE** and call it from the two specific functions (see sample API)
- If you really want a boost geometry type function which will take anything and try to run an algorithm on it (**NOT RECOMMENDED**), make it templatized in the header
- As of writing within/covered_by and a few other algorithms don't define functions taking GridPointRing and GridPointPoly since it's not needed at the moment, even though it could (and probably should) be added

Utilities for writing generalized algorithms

- Get_x and get_y for points (legacy point objects have different ways of accessing x and y)
- Typedef for child geometries (e.g., ring_type in Polygon)
    - Useful if you want to add a ring to a generalized Polygon or something along those lines

```
template <typename Poly>
void frankenstein(Poly* poly)
{
    typename Poly::point_type point(get_x((*poly).outer()[0]) + 5,
                                    get_y((*poly).outer()[0]));
    cout << boost::geometry::dsv(point) << endl;
    typename Poly::ring_type ring;
    boost::geometry::read_wkt("POLYGON((0␣0,␣0␣4,␣2␣0))", ring);
    poly->inners().push_back(std::move(ring));
}
...
GridLinePoly90 poly({GridLineRing90({{0,0}})});
frankenstein(&poly);
```

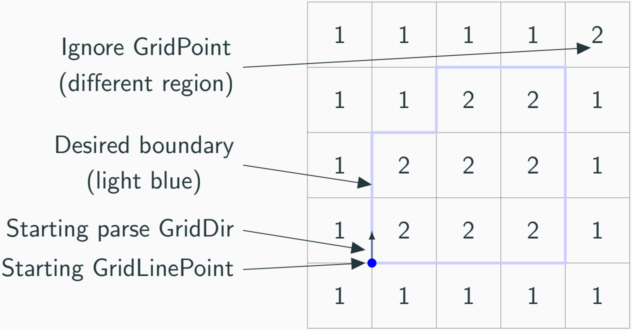
Boundary extraction

Requirements

- Should take DImage and find GridLineRings surrounding regions of GridPoints with the same label (called a **boundary**)
- The GridLineLineString sandwiched between two different-labeled regions should be returned as a **border** except when one side is a region of zeroes (obstacle)
- Regions within regions should become holes in the parent region's polygon (but also be returned as region boundaries)
- Obstacles should become holes, but not be returned as region boundaries or have borders

Extracting a boundary

- Start at lower-left hand GridLinePoint
- Start moving with GridDir::UP
- Want to trace out boundary clockwise
- Regions connected diagonally should be parsed as separate regions



Ignore GridPoint (different region)

Desired boundary (light blue)

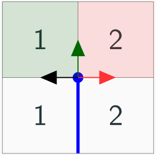
Starting parse GridDir

Starting GridLinePoint

1	1	1	1	2
1	1	2	2	1
1	2	2	2	1
1	2	2	2	1
1	1	1	1	1

Tracing the boundary

- Consider turning right
  - Check whether GridPoint to left of turn belongs to region
  - If not, make a right turn
  - If yes, continue to considering going straight
- Then consider going straight, then finally make a left turn if necessary
- Push back GridLinePoint to boundary whenever make a turn



Should pick the green arrow
since that's the first direction
without a 2 on the lefthand side

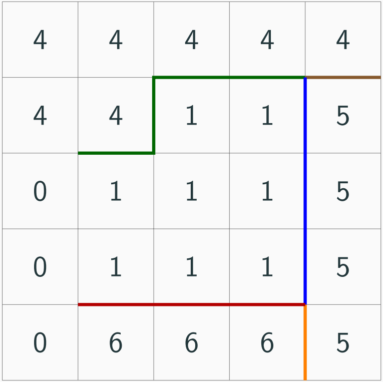
Keeping track of visited gridline connections

- After every traversal step:
    - Mark the GridLineConnector spanning two adjacent GridLinePoints as visited
    - Also mark the direction in which it was traversed
    - Each GridLineConnector can be visited up to twice (once from each side)

Extracting borders

- Can be done simultaneously with boundary traversal
- If the interior point is a zero (obstacle) or the GridLineConnector has been visited, don't extract border
- Otherwise, when the label on the outside changes, cap off the current border and start a new one
- When hit an obstacle or edge of image, cap off border but don't start new one
- Append to current border when turning

Example border parse result



4	4	4	4	4
4	4	1	1	5
0	1	1	1	5
0	1	1	1	5
0	6	6	6	5

Maintaining the boundary stack

- How do we know which regions are within other regions (should be holes)?
    - Sweep across DImage from left to right, row by row, maintaining a stack of regions
    - The sweep considers vertical GridLine**Connectors**, not GridPoints or GridLinePoints
    - Decide whether to pop, push, or create by checking the traversal direction and whether the right gridpoint's label is on the stack
    - If a new region is created and a previous region is on the stack, make the new region a hole in the previous one

Thank you!