CS296 Project Report
Group 18

Rohit Kumar
Roll No 120050028
rohit@cse.iitb.ac.in

Suman Sourabh
Roll No 120050031
sumansourabh26@cse.iitb.ac.in

Nitin Chandrol
Roll No 120050035
ntnchandrol@cse.iitb.ac.in

November 28, 2014

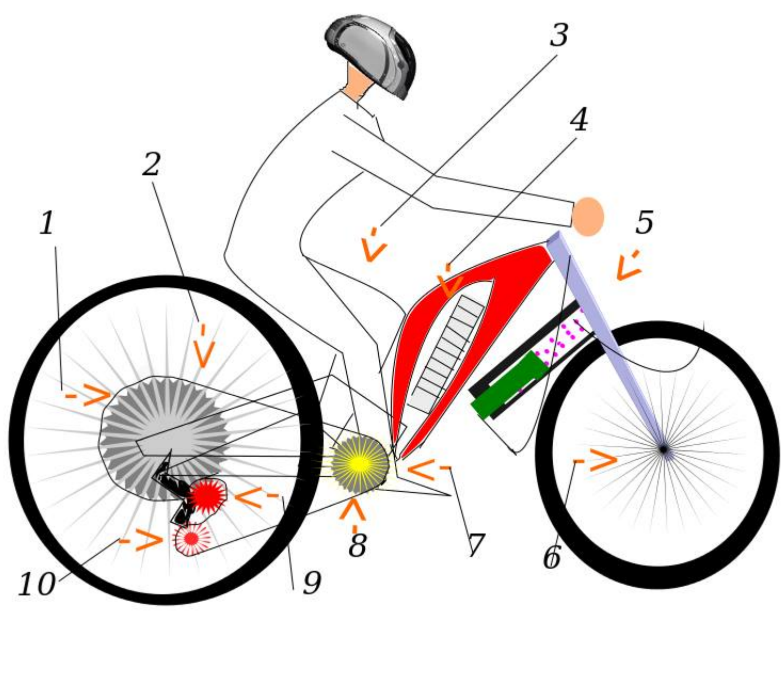
1 Introduction

The purpose of this report is to explain the simulation of important parts of cycle like chain motion, pedal motion, front and rear wheel motion and the complex gear changing mechanism.

Report analyses the relation between step time, collision time, velocity update, position update, loop time with iteration number graphically and it also describes reasoning behind such plot behaviours. Along with plots optimization for better performance is done by changing the simulation parameters like length of chain and friction coefficient. Another optimization is done by using -On flags. While profiling through callgraph we figure out which are most dominant function calls and needs to be optimized. At last we concluded about interesting features that we encountered in the simulations.

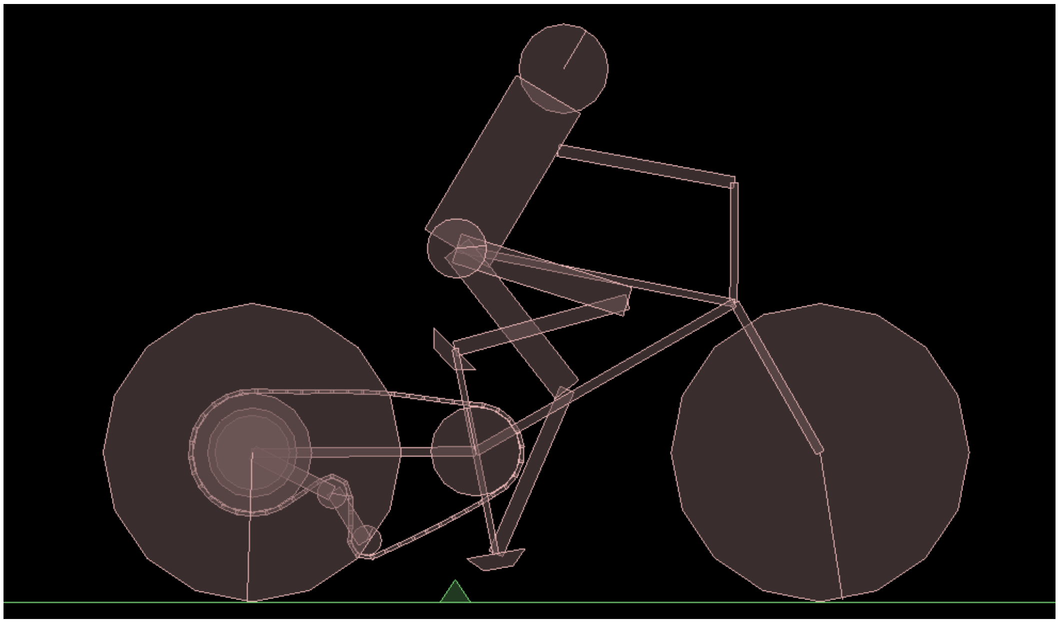
2 An overview of our final design

This is what we proposed at the beginning :



Our proposed design

Our final design:



Our final finished design

Following subsections explains in detail, different blocks of the bicycle using laws and equations of physics:

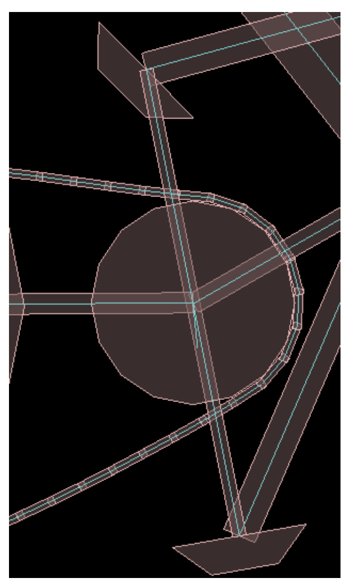
2.1 Pedal Wheel

This section works as main simulation initiator of whole cycle simulation. Its similar to real world cycle motion in which driver pushes pedals to provide angular velocity to pedal axle which further initiates motion of chain due to high friction coefficient between chain element and pedal axle. Motion is governed by the equation relating angular impulse to change in angular velocity.

$$I(\vec{w_f} - \vec{w_i}) = \Gamma\Delta t \qquad (1)$$

Here,

- $\mathrm{w_i}$ is the initial angular velocity of pedal axle
- $\mathrm{w_f}$ is the final angular velocity of pedal axle after a continous torque is appiled for Δt time.
- I is the moment of inertia of padel-axle about its center



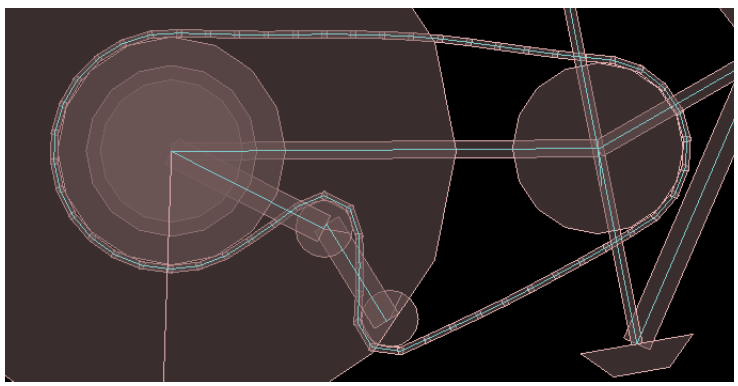
Screen shot of pedal wheel

- Γ is the torque applied to pedal axle

- Δt is the time period for which torque is applied

2.2 Chain System

Chain works as a connector between rear axles and pedal axle. It is made by joining numerous rectangular blocks which rolls smoothly over different axles and small pullies.Now as impulse by driver initiates angular motion in pedal,high coefficient of friction prevents slipping between chain elements and pedal axle and friction provides sufficient torque about pedal axle center such that chain starts rolling over the axles and movable pullies. To ensure rigidity and minimum slack in chain, density of chain elements is assigned a high value. Motion is governed by the equation relating torque and angular acceleration.



Screen shot of pedal wheel

$$\vec{r} * \vec{F} = I\alpha \tag{2}$$

Here,

- r is the radius vector of point of application of friction force.

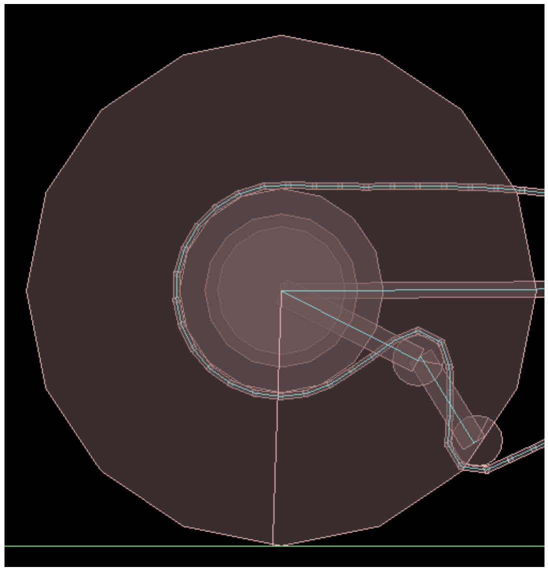
- F is the friction force between pedal-axle and chain element.

- I is moment of inertia of chain about pedal-axle center.

- α is angular acceleration of chain.

2.3 Rear Axle and Wheel Motion

Rear axle motion is initated by the motion of chain.As chain starts smooth rolling over pedal axle,friction between gear axle and chain provides angular velocity to gear axle in order to prevent slipping between chain and gear axle.Thus a angular impulse to pedal axle starts angular motion in rear gear to ensure that chain always remains tight.As gear axle and rear wheel are constrained by joints such that both moves with same angular velocity. Now friction between ground and rear wheel initiates combined translational and rotaional motion. Equations governing the motion are :



Screen shot of rear wheel

$$\vec{r} * \vec{F_a} - \vec{R} * \vec{F_g} = I_r \vec{\alpha} \tag{3}$$

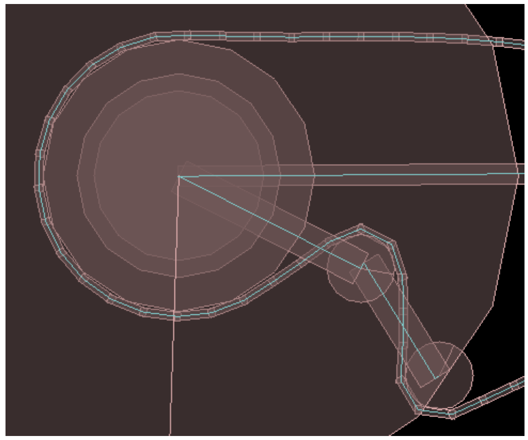
$$\vec{F_a} = m_r \vec{a_r} \tag{4}$$

Here,

- r is the radius vector of point of application of friction force between gear-axle and chain elements.
- F_a is the friction force between gear-axle and chain element.
- R is the radius vector of point of application of friction force between ground and rear-wheel.
- F_g is the friction force between ground and rear-wheel.
- I_r is moment of inertia of rear wheel and axle part.
- α is angular acceleration of rear wheel.
- m_r is the mass of rear wheel and gear axle.
- a_r is the acceleration of rear wheel.

2.4 Gear Mechanism

Gear Mechanism is the most complex part of the simulation.Gear Mechanism changes the radius of gear-axle.While decreasing gear we simply destroyed the outter gear axle, now chain gets reshaped with smaller radius axle. To adjust the change in length a structure of 2 rods and 2 small pullies is attached. This structure is hinged about rear-wheel center and free to move about it , so while changing gear it automatically gets reshaped to ensure minimum slack in chain.



Screen shot of the gear system

2.5 Front Wheel

Front wheel motion is initated due to rigid structure between front part and rear part of cycle.Due to translational motion of rear wheel, hinge at front wheel provide a forward impusle to iniate translational motion of front wheel.Now to avoid slipping between front wheel and ground friction force at ground provides sufficient torque.

$$\vec{R} * \vec{F_g} = I_f \vec{\alpha} \tag{5}$$

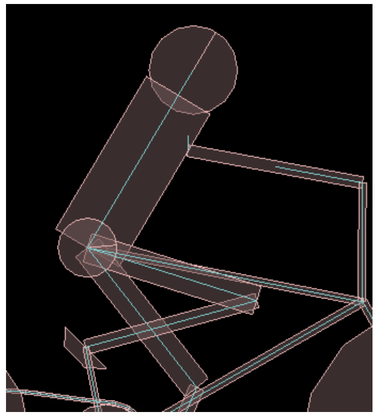
$$\vec{F_h} - \vec{F_g} = m_f \vec{a_f} \tag{6}$$

Here,

- R is the radius vector of point of application of friction force between ground and front-wheel.
- F_g is the friction force between ground and front-wheel.
- I_f is moment of inertia of front wheel.
- α is angular acceleration of front wheel.
- m_f is the mass of front wheel.
- a_f is the acceleration of front wheel.

2.6 Body Parts & Suspension

Main body parts like leg and thigh are joined with revolute joints with proper upper and lower limit to have realistic simulation. To provide support to whole body, body is attached with different rods using distancejoints.Here distancejoint plays an important role in suspension by using dampingratio and frequency features in distancejoints so whenever a sudden jerk comes bodies occur oscillatory motion with a dampening effect.



Screen shot of driver human body

3 Limitations in simulation

3.1 Brakes

In initial design we proposed a hydraulic break mechanism, but due to complexity of cycle simulation and gear mechanism we prefered to use disc break which provides an impusle to front wheel in opposite to direction of current motion.

3.2 3-D Gearing Mechanism

In realistic world a 3 dimentional motion of chain ensures the gear change mechanism with minimum slack.Here we tried a similar kind of simulation within 2 dimensional constraint of box2d.

4 Plot Analysis of the code

For analysing code for using it for optimization etc., we stopped the GLUI and GLUT calls. Since, the mechanism of our cycle was moslty keyboard oriented, we set the angular velocity and the linear velocity of the bicycle to a constant and generated csv data containing various time involved in the process. The data was analysed using matplotlib[3].

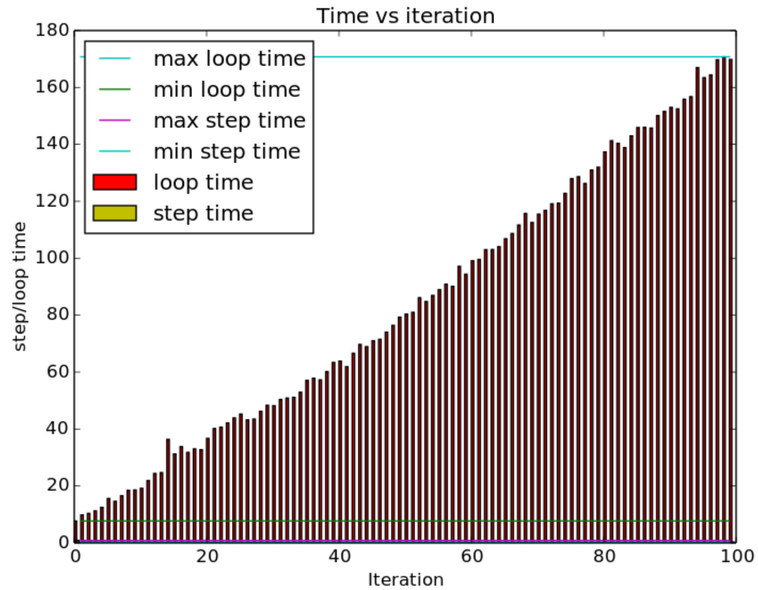
4.1 Plot 1: Loop time and average step time vs iteration values

Loop time increases with the number of iterations. As iteration number increases there will be more number of for loop execution and with each for loop initiating 150 steps.

$$t_{step} = \frac{x + y * itr}{itr} = \frac{x}{itr} + y \tag{7}$$

- Here, t_{step}, the average step is decreasing function of iteration value and for large value of iteration it gets stabilized.

Average step time decreases with number of iteration and gets stabilized after some reruns. Note that the average step time is not visible is the figure below as it has much lesser value than the loop time. The average step time is clearly visible in the next section.

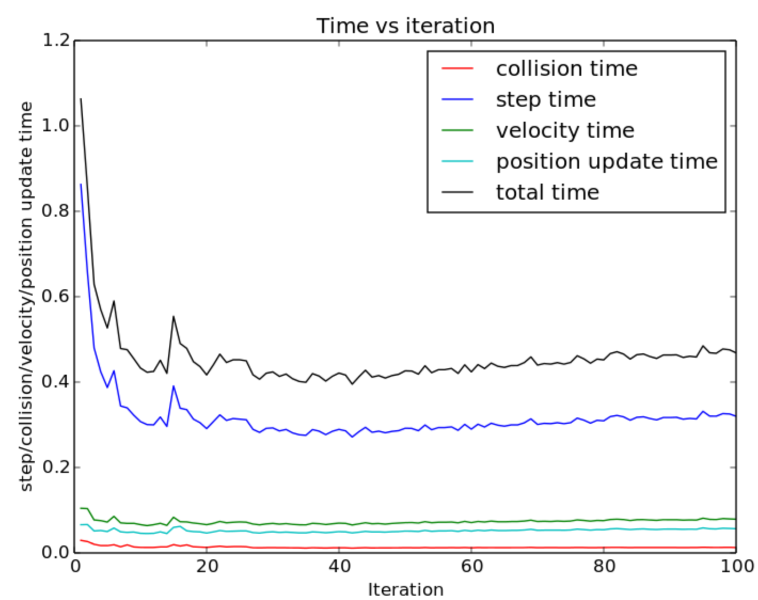


Loop time and average step time vs iteration values

4.2 Plot 2: Average step, collision, velocity and position update times vs iteration values

Position update,velocity update and collision time shows similar nature like step time and gradually decreases with time. Sum of all three update times contributes to a significant part of step time

These observations are clear from the following plot graphs:



Average step, collision, velocity, position and loop time vs iteration value

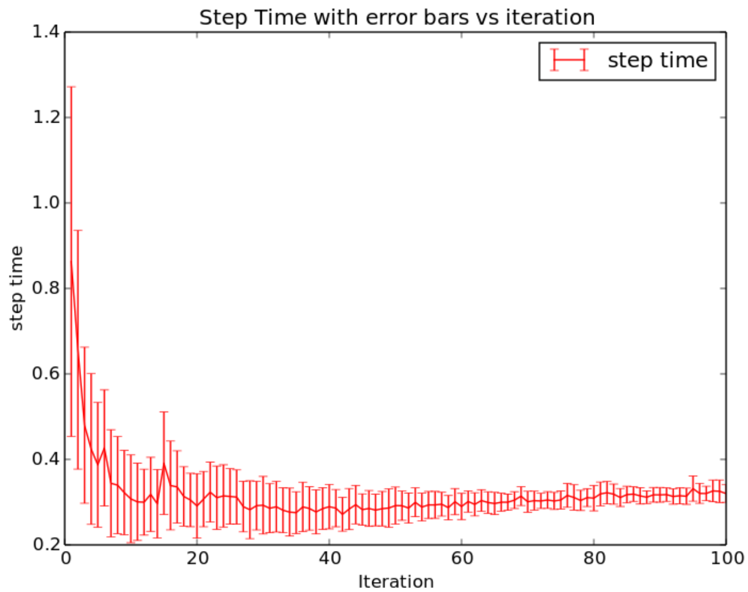
4.3 Plot3 : Average step time with error bars v/s iteration value

Plot step time line graph with error bars and deviation as maximum and minimum value for all reruns in particular iteration Deviation keeps on gradually decreasing with increase in iteration.

This is because:-

$$error = \Delta\frac{x + y * iter}{itr} = \frac{\Delta x}{itr} + \Delta y \tag{8}$$

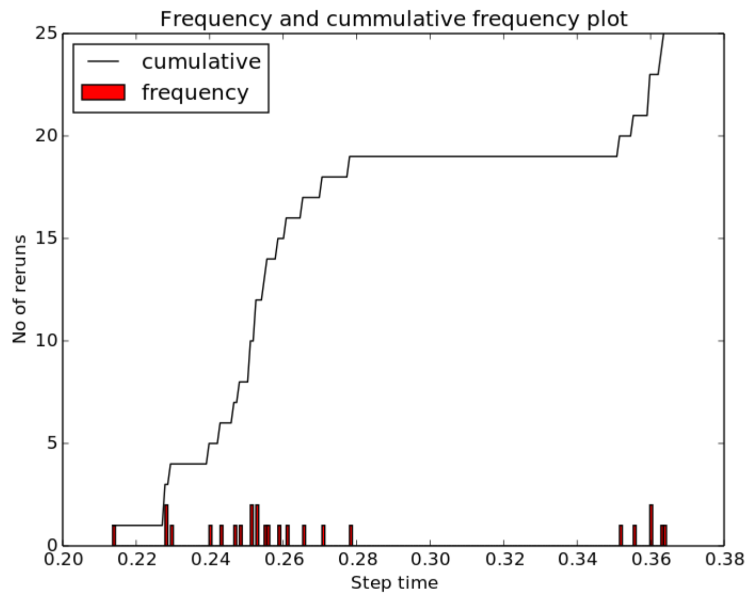
Now as the value of itr increases, the error decreases. This is described in the plot below:



Average step time with error bars v/s iteration value

4.4 Plot4 : Frequency and Cumulative Frequency Plot of Step Time

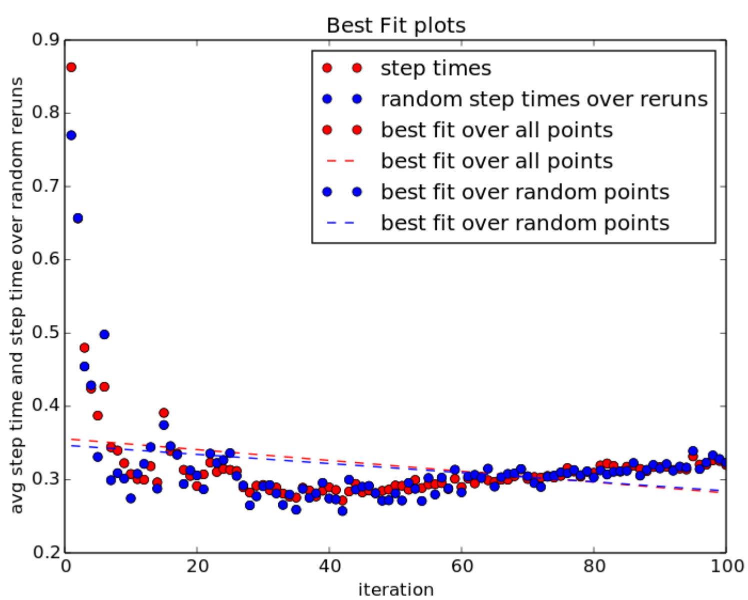
From the frequency plot of the step time, the most probable step time value (x-axis) can be found (by the plot below)



Frequency and Cumulative Frequency Plot of Step Time

4.5 Plot 5 : Best Linear fit for Step Time vs Iteration Value

As the step time varies slightly for all reruns for a particular iteration value thus the random points "blue" and average point "red" become close to each other (with slight deviation in slopes of best fit line).



Best Linear fit for Step Time vs Iteration Value

5 Profiling and Optimization

5.1 Step I :- Optimizing functions that take large time

With the help of profiling data generated by gprof, we identified[2] the functions which were taking large time. These functions in our simulation were *b2ContactSolver::SolveVelocityConstraints()* and *b2RevoluteJoint:: SolvePositionConstraints(b2SolverData const&)*. These functions were mostly used by the chain and pedal system as the chain joints were made using Revolute joints. So, we changed various features related to them like the length of a chain element and the friction coefficient of the chain.

5.1.1 Optimization by varying the size of the chain element

The chain is made of various small elements and joined by Revolute joints. Initially we kept the length of a chain element as *1.0f*.

As shown in Figure 1, the b2ContactSolver function takes the major chunk of time and also, there are a lot functions taking larger time.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 14.82      0.04     0.04                             b2ContactSolver::SolveVelocityConstraints()
 14.82      0.08     0.04                             void b2DynamicTree::Query<b2BroadPhase>(b2BroadPhase*, b2AABB const&) const
 11.11      0.11     0.03                             b2DynamicTree::InsertLeaf(int)
 11.11      0.14     0.03                             b2ContactManager::Collide()
 11.11      0.17     0.03                             b2World::DrawShape(b2Fixture*, b2Transform const&, b2Color const&)
  7.41      0.19     0.02                             b2ContactManager::AddPair(void*, void*)
  3.70      0.20     0.01                             b2PairLessThan(b2Pair const&, b2Pair const&)
  3.70      0.21     0.01                             void b2BroadPhase::UpdatePairs<b2ContactManager>(b2ContactManager*)
```

Figure 1: Profiling Data with chain element length as 1.0f

We reduced the chain length to 0.6f and got a lot of improvements(Figure 2). Not only the above two described functions were taking lesser time but the overall time taken by functions also got reduced.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 25.00      0.02     0.02                             b2DynamicTree::InsertLeaf(int)
 25.00      0.04     0.02                             b2World::DrawShape(b2Fixture*, b2Transform const&, b2Color const&)
 12.50      0.05     0.01                             b2DynamicTree::Balance(int)
 12.50      0.06     0.01                             b2ContactSolver::SolveVelocityConstraints()
 12.50      0.07     0.01                             b2DistanceJoint::InitVelocityConstraints(b2SolverData const&)
 12.50      0.08     0.01                             b2Island::Solve(b2Profile*, b2TimeStep const&, b2Vec2 const&, bool)
  0.00      0.08     0.00   258000     0.00     0.00  debug_draw_t::DrawSegment(b2Vec2 const&, b2Vec2 const&, b2Color const&)
  0.00      0.08     0.00    76000     0.00     0.00  debug_draw_t::DrawSolidPolygon(b2Vec2 const*, int, b2Color const&)
```

Figure 2: Profiling Data with chain element length as 0.6f

Finally we tried with chain element length as 0.4f. This resulted in an increase in overall time of the simulation and the time per call of the *debug_draw_t::DrawSolidPolygon* function also got increased.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ns/call  ns/call  name
 12.50      0.02     0.02                             void b2BroadPhase::UpdatePairs<b2ContactManager>(b2ContactManager*)
 12.50      0.04     0.02                             b2ContactSolver::SolveVelocityConstraints()
 12.50      0.06     0.02                             b2Body::SynchronizeFixtures()
  6.25      0.07     0.01    76000   131.59   131.59  debug_draw_t::DrawSolidPolygon(b2Vec2 const*, int, b2Color const&)
  6.25      0.08     0.01                             b2TimeOfImpact(b2TOIOutput*, b2TOIInput const*)
  6.25      0.09     0.01                             b2BroadPhase::QueryCallback(int)
  6.25      0.10     0.01                             b2DynamicTree::RemoveLeaf(int)
  6.25      0.11     0.01                             b2ContactSolver::b2ContactSolver(b2ContactSolverDef*)
  6.25      0.12     0.01                             b2RevoluteJoint::SolvePositionConstraints(b2SolverData const&)
```

Figure 3: Profiling Data with chain element length as 0.4f

So, finally we kept the chain length as 0.6f for optimal results.

5.1.2 Optimization by adjusting the friction values between various elements

The second significant change that was observed was when the friction value of the chain elements was changed from 10.f (initial value) till 30.0f (final value). A decrease in cumulative time of all functions was observed to decrease from 0.27 units to 0.21 units. Also the time required by above two described functions decreased.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 22.22      0.06     0.06                             b2ContactSolver::SolveVelocityConstraints()
 14.82      0.10     0.04                             b2RevoluteJoint::SolveVelocityConstraints(b2SolverData const&)
 11.11      0.13     0.03                             b2RevoluteJoint::SolvePositionConstraints(b2SolverData const&)
  7.41      0.15     0.02                             b2Distance(b2DistanceOutput*, b2SimplexCache*, b2DistanceInput const*)
  7.41      0.17     0.02                             b2DynamicTree::InsertLeaf(int)
  7.41      0.19     0.02                             b2World::SolveTOI(b2TimeStep const&)
  7.41      0.21     0.02                             b2Mat33::Solve22(b2Vec2 const&) const
  3.70      0.22     0.01                             b2WeldJoint::SolveVelocityConstraints(b2SolverData const&)
  3.70      0.23     0.01                             b2DynamicTree::Balance(int)
  3.70      0.24     0.01                             b2ContactSolver::SolvePositionConstraints()
  3.70      0.25     0.01                             b2RevoluteJoint::InitVelocityConstraints(b2SolverData const&)
  3.70      0.26     0.01                             void b2DynamicTree::Query<b2BroadPhase>(b2BroadPhase*, b2AABB const&) const
  3.70      0.27     0.01                             b2Body::ShouldCollide(b2Body const*) const
  0.00      0.27     0.00   252000     0.00     0.00  debug_draw_t::DrawSegment(b2Vec2 const&, b2Vec2 const&, b2Color const&)
  0.00      0.27     0.00    74000     0.00     0.00  debug_draw_t::DrawSolidPolygon(b2Vec2 const*, int, b2Color const&)
  0.00      0.27     0.00    43730     0.00     0.00  b2ContactListener::PreSolve(b2Contact*, b2Manifold const*)
  0.00      0.27     0.00    43730     0.00     0.00  b2ContactListener::PostSolve(b2Contact*, b2ContactImpulse const*)
  0.00      0.27     0.00    10000     0.00     0.00  debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&, b2Color const&)
  0.00      0.27     0.00      225     0.00     0.00  b2ContactListener::BeginContact(b2Contact*)
  0.00      0.27     0.00      189     0.00     0.00  b2ContactListener::EndContact(b2Contact*)
  0.00      0.27     0.00        3     0.00     0.00  _init
  0.00      0.27     0.00        1     0.00     0.00  cs296::base_sim_t::base_sim_t()
  0.00      0.27     0.00        1     0.00     0.00  cs296::dominos_t::dominos_t()
  0.00      0.27     0.00        1     0.00     0.00  main
```

Figure 4: Profiling Data with friction value 0f chain as 10.0f

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 22.22      0.06     0.06                             b2ContactSolver::SolveVelocityConstraints()
 14.82      0.10     0.04                             b2RevoluteJoint::SolveVelocityConstraints(b2SolverData const&)
 11.11      0.13     0.03                             b2RevoluteJoint::SolvePositionConstraints(b2SolverData const&)
  7.41      0.15     0.02                             b2Distance(b2DistanceOutput*, b2SimplexCache*, b2DistanceInput const*)
  7.41      0.17     0.02                             b2DynamicTree::InsertLeaf(int)
  7.41      0.19     0.02                             b2World::SolveTOI(b2TimeStep const&)
  7.41      0.21     0.02                             b2Mat33::Solve22(b2Vec2 const&) const
  3.70      0.22     0.01                             b2WeldJoint::SolveVelocityConstraints(b2SolverData const&)
  3.70      0.23     0.01                             b2DynamicTree::Balance(int)
  3.70      0.24     0.01                             b2ContactSolver::SolvePositionConstraints()
  3.70      0.25     0.01                             b2RevoluteJoint::InitVelocityConstraints(b2SolverData const&)
  3.70      0.26     0.01                             void b2DynamicTree::Query<b2BroadPhase>(b2BroadPhase*, b2AABB const&) const
  3.70      0.27     0.01                             b2Body::ShouldCollide(b2Body const*) const
  0.00      0.27     0.00   252000     0.00     0.00  debug_draw_t::DrawSegment(b2Vec2 const&, b2Vec2 const&, b2Color const&)
  0.00      0.27     0.00    74000     0.00     0.00  debug_draw_t::DrawSolidPolygon(b2Vec2 const*, int, b2Color const&)
  0.00      0.27     0.00    43730     0.00     0.00  b2ContactListener::PreSolve(b2Contact*, b2Manifold const*)
  0.00      0.27     0.00    43730     0.00     0.00  b2ContactListener::PostSolve(b2Contact*, b2ContactImpulse const*)
  0.00      0.27     0.00    10000     0.00     0.00  debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&, b2Color const&)
  0.00      0.27     0.00      225     0.00     0.00  b2ContactListener::BeginContact(b2Contact*)
  0.00      0.27     0.00      189     0.00     0.00  b2ContactListener::EndContact(b2Contact*)
  0.00      0.27     0.00        3     0.00     0.00  _init
  0.00      0.27     0.00        1     0.00     0.00  cs296::base_sim_t::base_sim_t()
  0.00      0.27     0.00        1     0.00     0.00  cs296::dominos_t::dominos_t()
  0.00      0.27     0.00        1     0.00     0.00  main
```

Figure 5: Profiling Data with friction value 0f of chain as 30.0f

5.2 Step II : -On options

The second step that we took towards optimization is invoking the various optimization flags (-On) that are offered by gnu g++ compiler. We tried the -On option with n varing from 1 to 4.

As described by the profile data shown in Figure 1,2,3 and 4, we took the following inferences from it :

1. The cumlative time was least for -O2 option (0.16 units) as compared to that of -O1(0.22 units), -O3(0.28 units) and -O4(0.27 units). Here 1 unit is 0.01 seconds
2. The most time taking functions i.e. *b2ContactSolver::SolveVelocityConstraints() and b2RevoluteJoint:: SolvePositionConstraints(b2SolverData const&)* were taking least time with -O2 optimization flag (0.04 units for both) as compared to -O3 (0.06 units), -O4 (0.10 units) and -O1 (0.06 units)

Following these observations we took the -O2 flag as the optimal flag for our simulation.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 31.82      0.07     0.07                             b2ContactSolver::SolveVelocityConstraints()
 13.64      0.10     0.03                             b2RevoluteJoint::SolvePositionConstraints(b2SolverData const&)
  9.09      0.12     0.02                             b2CollidePolygonAndCircle(b2Manifold*, b2PolygonShape const*, b2Transform const&,
b2CircleShape const*, b2Transform const&)
  9.09      0.14     0.02                             b2RevoluteJoint::SolveVelocityConstraints(b2SolverData const&)
  4.55      0.15     0.01                             b2TimeOfImpact(b2TOIOutput*, b2TOIInput const*)
  4.55      0.16     0.01                             b2WeldJoint::SolveVelocityConstraints(b2SolverData const&)
  4.55      0.17     0.01                             b2DynamicTree::InsertLeaf(int)
  4.55      0.18     0.01                             b2ContactFilter::ShouldCollide(b2Fixture*, b2Fixture*)
  4.55      0.19     0.01                             b2ContactSolver::SolvePositionConstraints()
  4.55      0.20     0.01                             b2World::Solve(b2TimeStep const&)
  4.55      0.21     0.01                             b2World::SolveTOI(b2TimeStep const&)
  4.55      0.22     0.01                             b2Fixture::Synchronize(b2BroadPhase*, b2Transform const&, b2Transform const&)
  0.00      0.22     0.00   258000     0.00     0.00  debug_draw_t::DrawSegment(b2Vec2 const&, b2Vec2 const&, b2Color const&)
  0.00      0.22     0.00    76000     0.00     0.00  debug_draw_t::DrawSolidPolygon(b2Vec2 const*, int, b2Color const&)
  0.00      0.22     0.00    43664     0.00     0.00  b2ContactListener::PreSolve(b2Contact*, b2Manifold const*)
  0.00      0.22     0.00    43664     0.00     0.00  b2ContactListener::PostSolve(b2Contact*, b2ContactImpulse const*)
  0.00      0.22     0.00    10000     0.00     0.00  debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&, b2Color const&
  0.00      0.22     0.00     1000     0.00     0.00  cs296::base_sim_t::step(cs296::settings_t*)
  0.00      0.22     0.00      361     0.00     0.00  b2ContactListener::BeginContact(b2Contact*)
  0.00      0.22     0.00      325     0.00     0.00  b2ContactListener::EndContact(b2Contact*)
  0.00      0.22     0.00        2     0.00     0.00  cs296::dominos_t::dominos_t()
  0.00      0.22     0.00        1     0.00     0.00  cs296::base_sim_t::draw_title(int, int, char const*)
  0.00      0.22     0.00        1     0.00     0.00  cs296::base_sim_t::base_sim_t()
  0.00      0.22     0.00        1     0.00     0.00  cs296::callbacks_t::single_step_cb(int)
  0.00      0.22     0.00        1     0.00     0.00  cs296::dominos_t::create()
  0.00      0.22     0.00        1     0.00     0.00  main
```

Figure 4: Profile data with -O1 option

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ns/call  ns/call  name
 12.50      0.02     0.02                             void b2BroadPhase::UpdatePairs<b2ContactManager>(b2ContactManager*)
 12.50      0.04     0.02                             b2ContactSolver::SolveVelocityConstraints()
 12.50      0.06     0.02                             b2Body::SynchronizeFixtures()
  6.25      0.07     0.01    76000   131.59   131.59  debug_draw_t::DrawSolidPolygon(b2Vec2 const*, int, b2Color const&)
  6.25      0.08     0.01                             b2TimeOfImpact(b2TOIOutput*, b2TOIInput const*)
  6.25      0.09     0.01                             b2BroadPhase::QueryCallback(int)
  6.25      0.10     0.01                             b2DynamicTree::RemoveLeaf(int)
  6.25      0.11     0.01                             b2ContactSolver::b2ContactSolver(b2ContactSolverDef*)
  6.25      0.12     0.01                             b2RevoluteJoint::SolvePositionConstraints(b2SolverData const&)
  6.25      0.13     0.01                             b2RevoluteJoint::SolveVelocityConstraints(b2SolverData const&)
  6.25      0.14     0.01                             void b2DynamicTree::Query<b2BroadPhase>(b2BroadPhase*, b2AABB const&) const
  6.25      0.15     0.01                             b2Body::ShouldCollide(b2Body const*) const
  6.25      0.16     0.01                             b2Mat33::Solve22(b2Vec2 const&) const
  0.00      0.16     0.00   258000     0.00     0.00  debug_draw_t::DrawSegment(b2Vec2 const&, b2Vec2 const&, b2Color const&)
  0.00      0.16     0.00    43664     0.00     0.00  b2ContactListener::PreSolve(b2Contact*, b2Manifold const*)
  0.00      0.16     0.00    43664     0.00     0.00  b2ContactListener::PostSolve(b2Contact*, b2ContactImpulse const*)
  0.00      0.16     0.00    10000     0.00     0.00  debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&, b2Color cons
  0.00      0.16     0.00      361     0.00     0.00  b2ContactListener::BeginContact(b2Contact*)
  0.00      0.16     0.00      325     0.00     0.00  b2ContactListener::EndContact(b2Contact*)
  0.00      0.16     0.00        3     0.00     0.00  _init
  0.00      0.16     0.00        1     0.00     0.00  cs296::base_sim_t::base_sim_t()
  0.00      0.16     0.00        1     0.00     0.00  cs296::dominos_t::dominos_t()
  0.00      0.16     0.00        1     0.00     0.00  main
```

Figure 5: Profile data with -O2 option

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ns/call  ns/call  name
 21.43      0.06     0.06                             b2ContactSolver::SolveVelocityConstraints()
 12.50      0.10     0.04                             b2RevoluteJoint::SolvePositionConstraints(b2SolverData const&)
  7.14      0.12     0.02                             b2World::DrawShape(b2Fixture*, b2Transform const&, b2Color const&)
  7.14      0.14     0.02                             b2PolygonShape::ComputeAABB(b2AABB*, b2Transform const&, int) const
  3.57      0.15     0.01    76000   131.58   131.58  debug_draw_t::DrawSolidPolygon(b2Vec2 const*, int, b2Color const&)
  3.57      0.16     0.01                             void b2BroadPhase::UpdatePairs<b2ContactManager>(b2ContactManager*)
  3.57      0.17     0.01                             b2DynamicTree::InsertLeaf(int)
  3.57      0.18     0.01                             b2ContactSolver::SolveTOIPositionConstraints(int, int)
  3.57      0.19     0.01                             b2ContactSolver::InitializeVelocityConstraints()
  3.57      0.20     0.01                             b2RevoluteJoint::SolveVelocityConstraints(b2SolverData const&)
  3.57      0.21     0.01                             b2BlockAllocator::Allocate(int)
  3.57      0.22     0.01                             b2ContactManager::AddPair(void*, void*)
  3.57      0.23     0.01                             b2ContactManager::Collide()
  3.57      0.24     0.01                             b2World::Solve(b2TimeStep const&)
  3.57      0.25     0.01                             b2Island::Solve(b2Profile*, b2TimeStep const&, b2Vec2 const&, bool)
  3.57      0.26     0.01                             void b2DynamicTree::Query<b2BroadPhase>(b2BroadPhase*, b2AABB const&) const
  3.57      0.27     0.01                             b2Mat33::Solve22(b2Vec2 const&) const
  3.57      0.28     0.01                             b2Timer::GetMilliseconds() const
  0.00      0.28     0.00   258000     0.00     0.00  debug_draw_t::DrawSegment(b2Vec2 const&, b2Vec2 const&, b2Color const&)
  0.00      0.28     0.00    43664     0.00     0.00  b2ContactListener::PreSolve(b2Contact*, b2Manifold const*)
  0.00      0.28     0.00    43664     0.00     0.00  b2ContactListener::PostSolve(b2Contact*, b2ContactImpulse const*)
  0.00      0.28     0.00    10000     0.00     0.00  debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&, b2Color const&)
  0.00      0.28     0.00      361     0.00     0.00  b2ContactListener::BeginContact(b2Contact*)
  0.00      0.28     0.00      325     0.00     0.00  b2ContactListener::EndContact(b2Contact*)
  0.00      0.28     0.00        3     0.00     0.00  _init
  0.00      0.28     0.00        1     0.00     0.00  cs296::base_sim_t::base_sim_t()
  0.00      0.28     0.00        1     0.00     0.00  cs296::dominos_t::dominos_t()
  0.00      0.28     0.00        1     0.00     0.00  main
```

Figure 6: Profile data with -O3 option

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 22.22      0.06     0.06                             void b2DynamicTree::Query<b2BroadPhase>(b2BroadPhase*, b2AABB const&) const
 18.52      0.11     0.05                             b2ContactSolver::SolveVelocityConstraints()
 18.52      0.16     0.05                             b2RevoluteJoint::SolveVelocityConstraints(b2SolverData const&)
  7.41      0.18     0.02                             b2ContactManager::AddPair(void*, void*)
  3.70      0.19     0.01                             b2Distance(b2DistanceOutput*, b2SimplexCache*, b2DistanceInput const*)
  3.70      0.20     0.01                             void b2BroadPhase::UpdatePairs<b2ContactManager>(b2ContactManager*)
  3.70      0.21     0.01                             b2DynamicTree::InsertLeaf(int)
  3.70      0.22     0.01                             b2RevoluteJoint::SolvePositionConstraints(b2SolverData const&)
  3.70      0.23     0.01                             b2WorldManifold::Initialize(b2Manifold const*, b2Transform const&, float, b2Transf
const&, float)
  3.70      0.24     0.01                             b2World::Solve(b2TimeStep const&)
  3.70      0.25     0.01                             b2PolygonShape::ComputeAABB(b2AABB*, b2Transform const&, int) const
  3.70      0.26     0.01                             b2Body::ShouldCollide(b2Body const*) const
  3.70      0.27     0.01                             b2Mat33::Solve22(b2Vec2 const&) const
  0.00      0.27     0.00   258000     0.00     0.00  debug_draw_t::DrawSegment(b2Vec2 const&, b2Vec2 const&, b2Color const&)
  0.00      0.27     0.00    76000     0.00     0.00  debug_draw_t::DrawSolidPolygon(b2Vec2 const*, int, b2Color const&)
  0.00      0.27     0.00    43664     0.00     0.00  b2ContactListener::PreSolve(b2Contact*, b2Manifold const*)
  0.00      0.27     0.00    43664     0.00     0.00  b2ContactListener::PostSolve(b2Contact*, b2ContactImpulse const*)
  0.00      0.27     0.00    10000     0.00     0.00  debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&, b2Color const&)
  0.00      0.27     0.00      361     0.00     0.00  b2ContactListener::BeginContact(b2Contact*)
  0.00      0.27     0.00      325     0.00     0.00  b2ContactListener::EndContact(b2Contact*)
  0.00      0.27     0.00        3     0.00     0.00  _init
  0.00      0.27     0.00        1     0.00     0.00  cs296::base_sim_t::base_sim_t()
  0.00      0.27     0.00        1     0.00     0.00  cs296::dominos_t::dominos_t()
  0.00      0.27     0.00        1     0.00     0.00  main
```

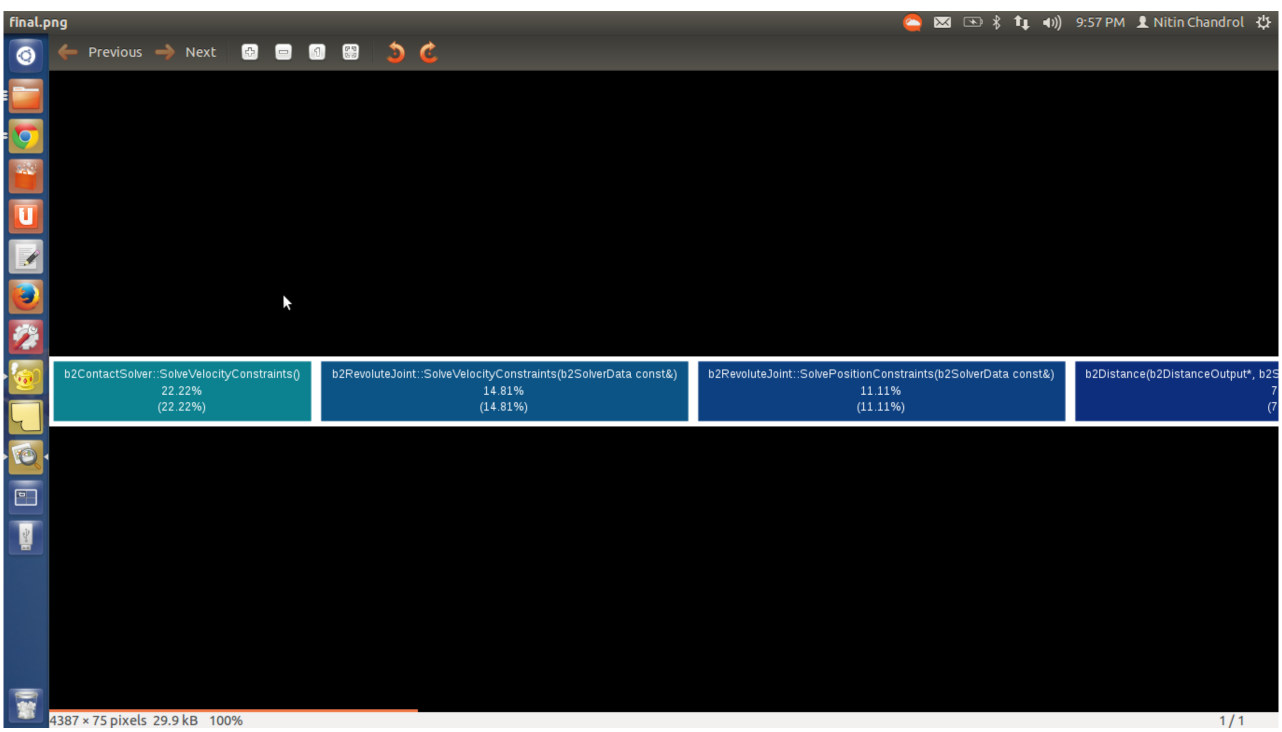
Figure 7: Profile data with -O4 option

5.3 Call graph using python script

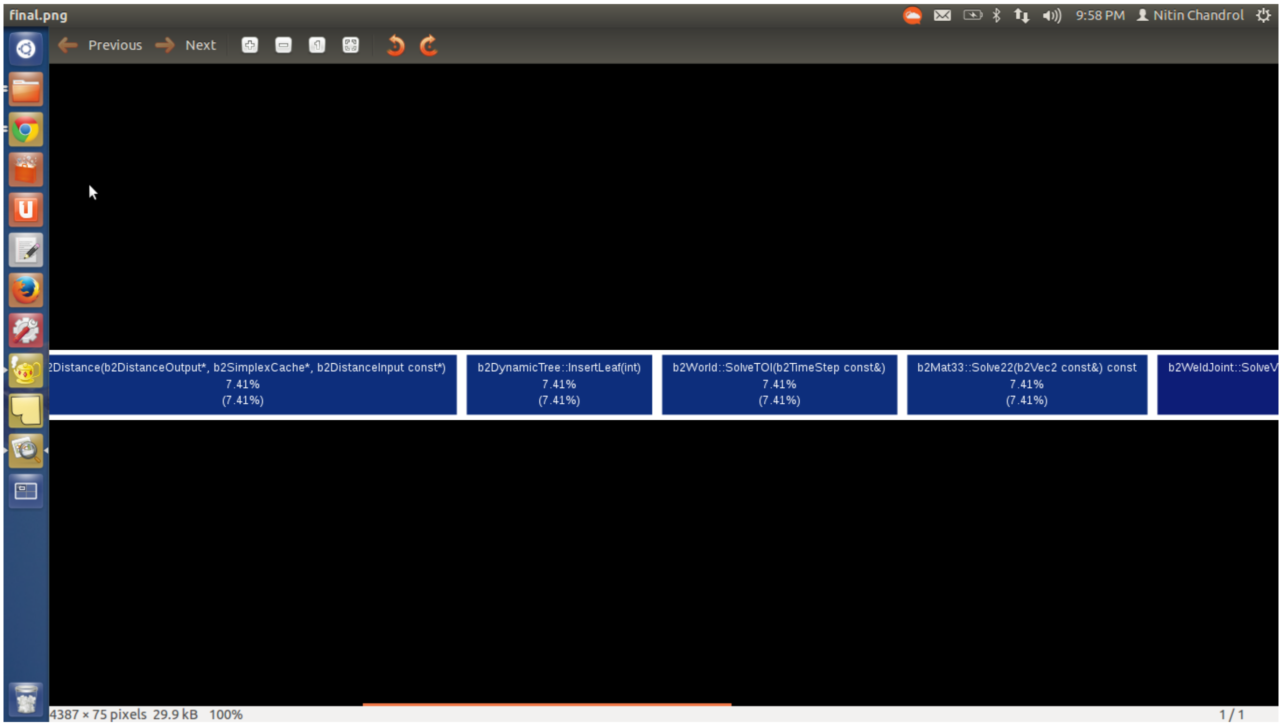
The callgraph of the final profile is given in the figure below. This is generated by the gprof2dot python script[1].

5.3.1 Debug Profile

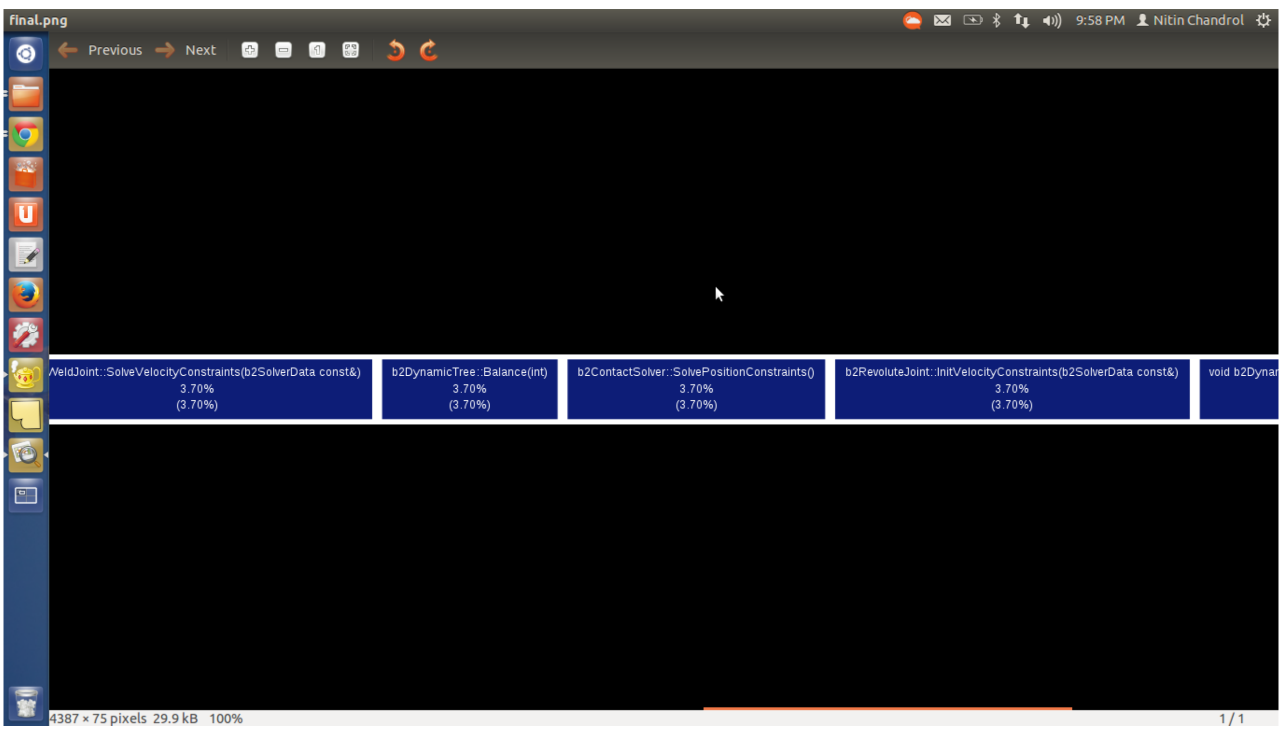
The call graphs generated in the debug profile are :



Part 1: Call Graph for Debug mode



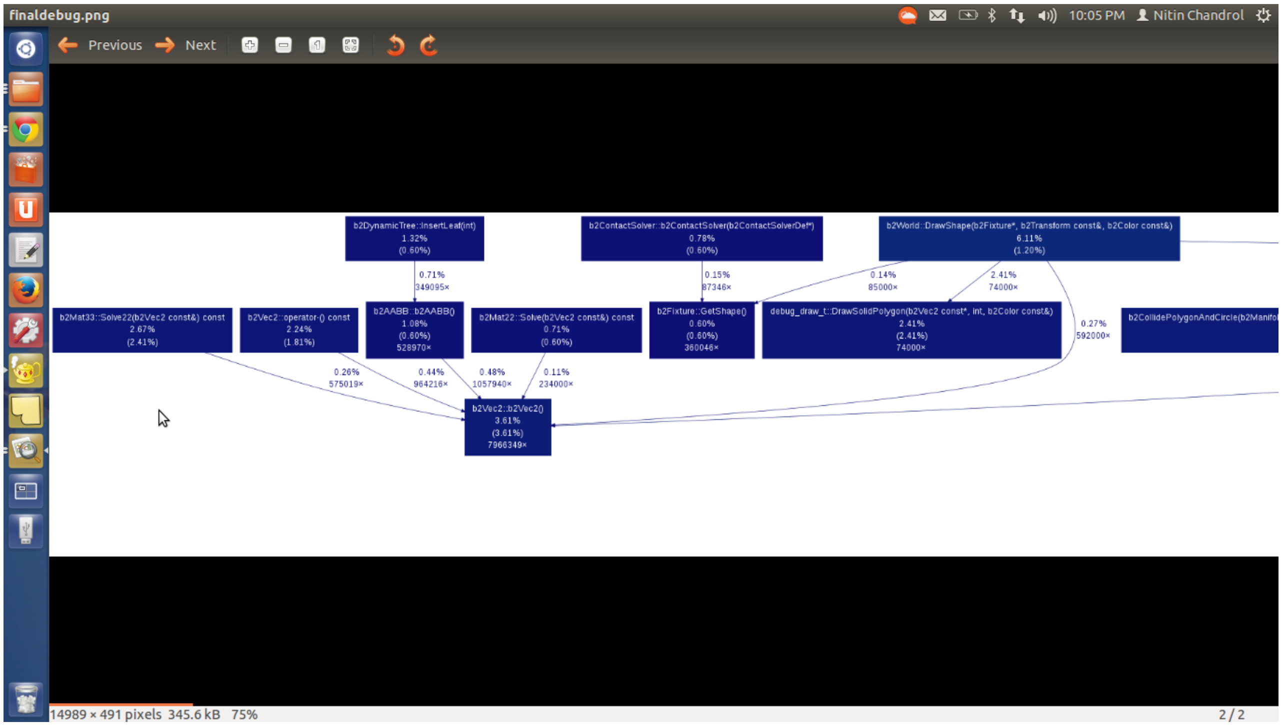
Part 2: Call Graph for Debug mode



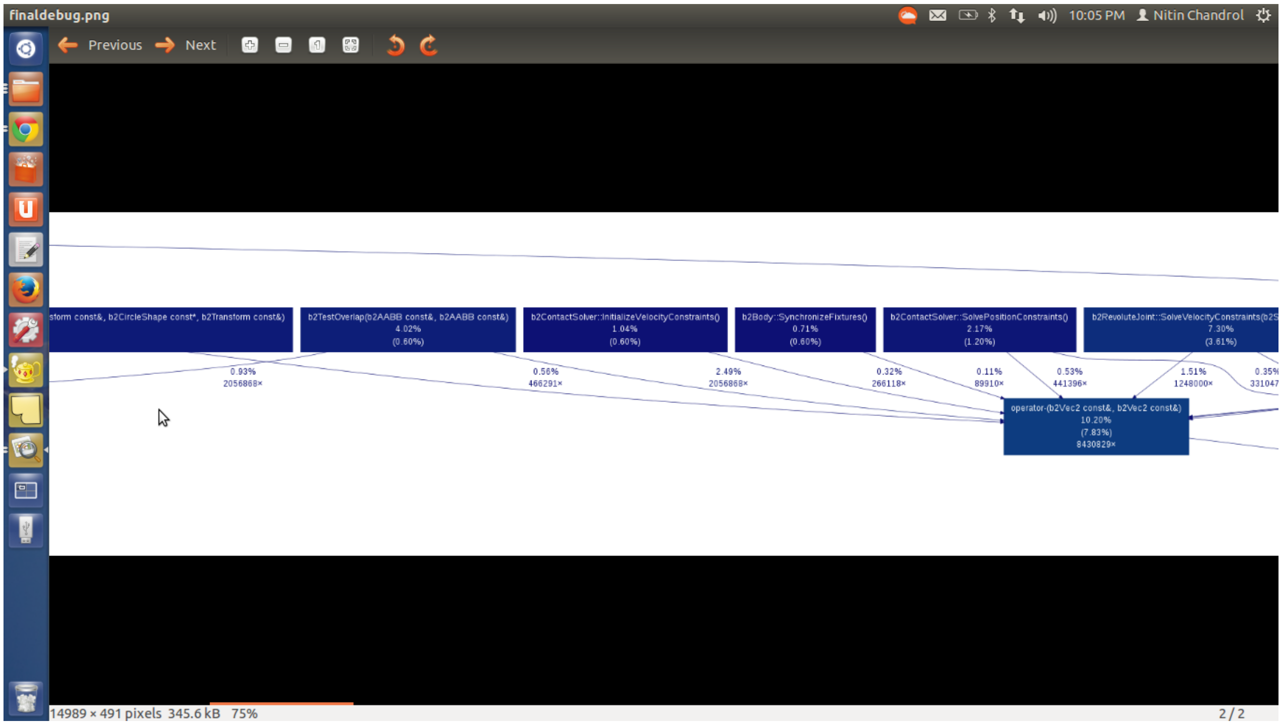
Part 3: Call Graph for Debug mode

5.3.2 Release Profile

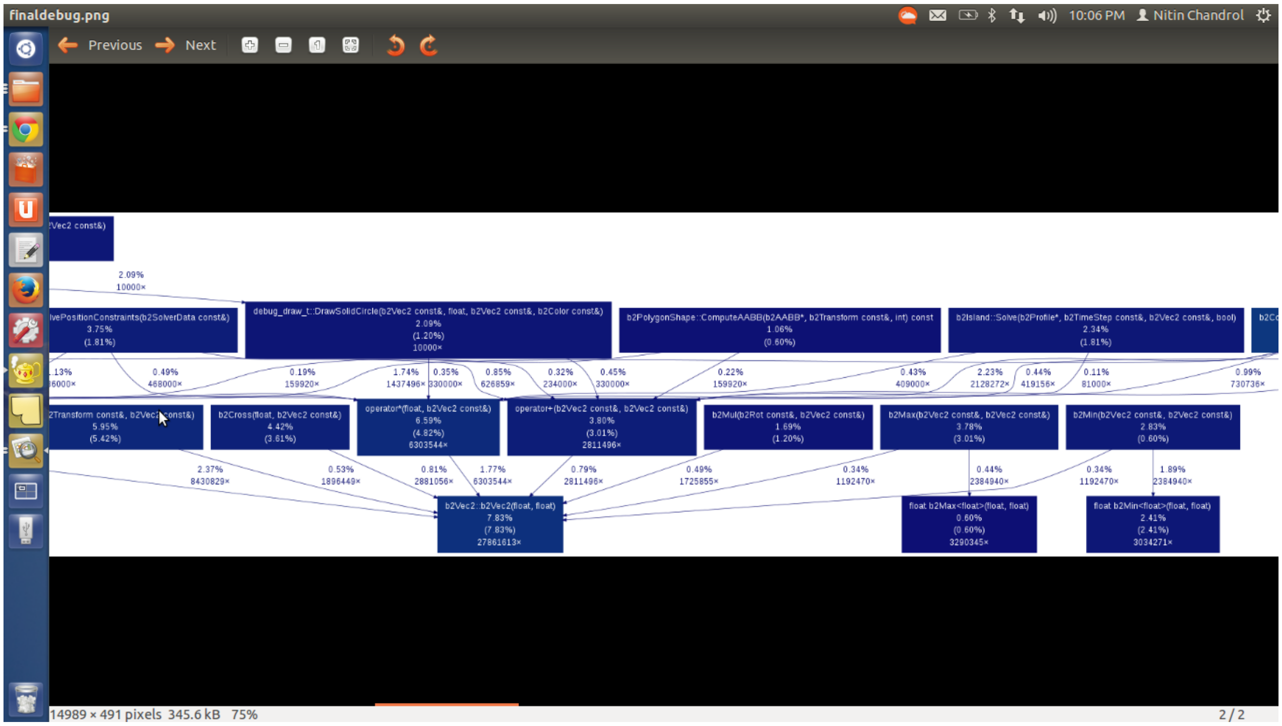
The call graphs generated in the release profile are :



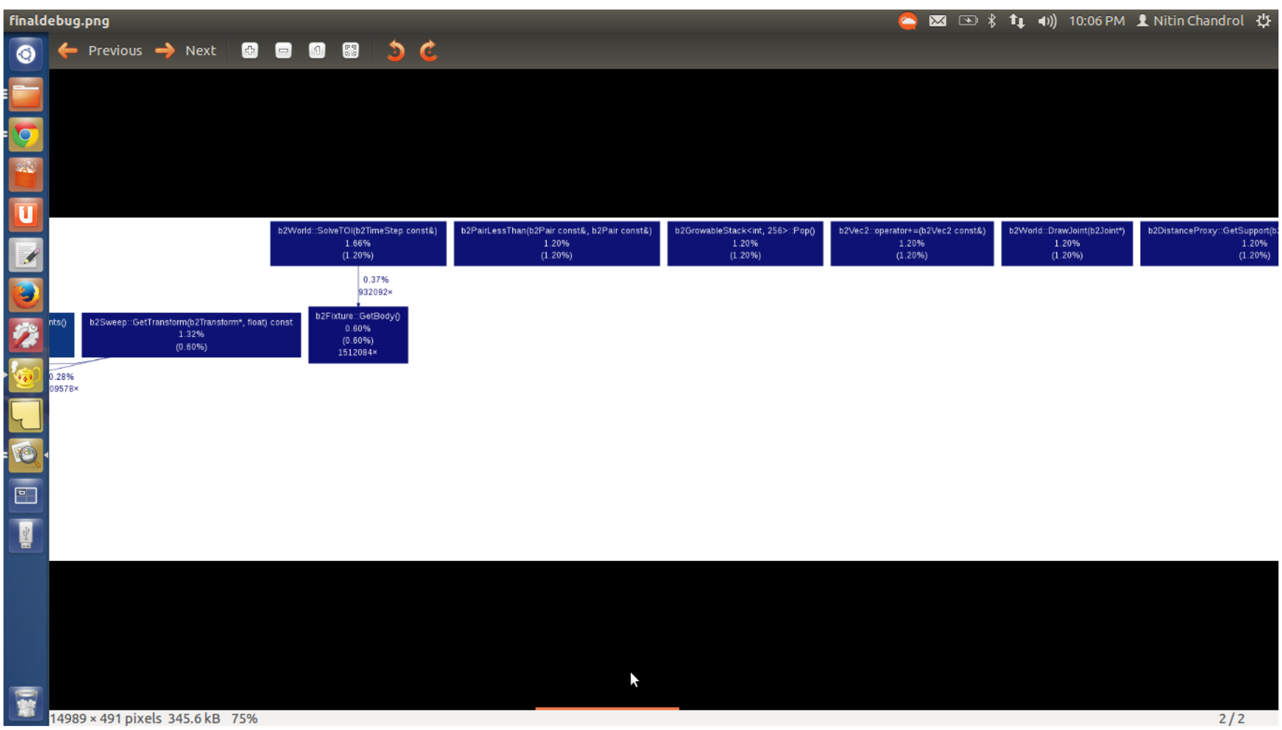
Part 1: Call Graph for Release mode



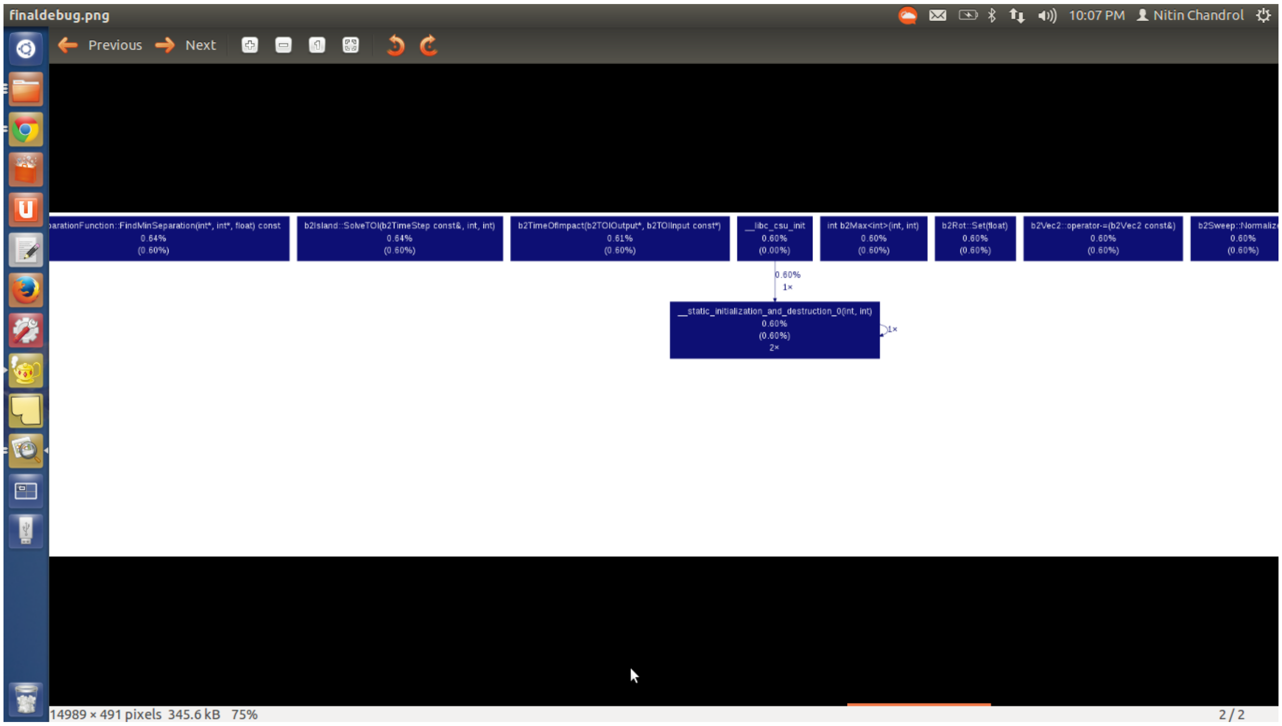
Part 2: Call Graph for Release mode



Part 3: Call Graph for Release mode



Part 4: Call Graph for Release mode



Part 5: Call Graph for Release mode

6 Interesting Features of design

Chain System One of the major problems we encountered was to build a tight chain structure (i.e. without slacks in between) as it had to take a circular shape. In this case we used the box2D property that two similar type of dynamic bodies (b2DynamicBody) collide by default and push away each other. Due to this property, the chain get automatically fits over the pedal and gear system although it is given a linear shape initially.

Gear Mechanism Most interesting and complex part of our simulation is gear mechanism, where in real-type we can change cycle gear and reshape rear-axle with the help of flexible rod and small pully structure. We succeeded in making a gear mechanism similar to real world by using rod and small pully structure.

Suspension DistanceJoint feature of Box2D simplified the suspension mechanism.We joined the driver body parts with cycle using distancejoints where using damping ratio and frequency property to ensure that driver undergoes dampening oscillatory motion under a sudden jerk and impulse.

7 Conclusions

We succeeded in simulating major part of our initial design with following all basic mechanics involved in a gear cycle. Major problems we faced while project were chain elements distortion while increasing angular impulse at pedal axle and choosing appropriate friction values between axle - chain and ground - wheel to attain sufficient velocity.

We discovered how to get time of particular process using different timers like time, gettimeoftheday and the difference between them. We also learnt to profile code and to analyse time taken by each function call individually and number of calls for a particular function using call graph.

Further we learnt to create and review call graph plots using python script and then analyze function call complexity graphically and learnt comparing each function call time to decide which part of code is required to be optimized.

References

[1] Jos Fonseca. https://code.google.com/p/jrfonseca/wiki/gprof2dot.

[2] Gnu Optimize options. http://gcc.gnu.org/onlinedocs/gcc/optimize-options.html.

[3] Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers. *http://www.openbookproject.net/thinkcs/python/english3e/*.