ATRテクニカルレポート表紙

[公開]

TR-M-0051

拍手音を用いた共感の念の創出

西村　竜一　　宮里　勉
Ryouichi NISHIMURA　　Tsutomu MIYASATO

2000.3.31

ATR 知能映像通信研究所

1　はじめに

遠隔会議や遠隔操作，あるいはエンターテイメントなどへの応用を目指した，仮想環境や仮想現実感に関する研究は，現在，新たなコミュニケーション形態の実現などへ向けて，今なお盛んに研究が進められている．聴覚的な仮想現実感に関しては，頭部伝達関数を利用した3次元立体音像に関する研究が広く知られている．これらの研究成果の一部は，実際にホームシアターなどで用いられる音響技術などと結び付き，実用化されるに至っている．このような仮想現実感を実現するための各種デバイスの開発や，信号処理技術の急速な発展は，今後も更に加速することが予想される．

一方で，最近になって，これら仮想現実感を創出する道具を用いて提示すべき，コンテンツの量的不足や，質の改善などが問題視されるようになってきた．今後さらに，問題になることが予想されるのは，いつでもどこでも好きなときに好きな (現実と区別が付かない) 情報を得ることができるという状況は，個人の選択肢を飛躍的に増大させてくれるものではあるが，逆に，現実社会において社会性の欠如した人間を生みだす危険性をもはらんでいるということである．

この問題に対し，現在の実世界では，人は通常多くの他人と同じ「場」を共有しているため，ある場面に直面した時に取るべき行動を他人から学ぶことができる．また，それと同時に，自分がとった行動が直ちに他人からの批評の対象に曝されるため，常に自制心が働く．「ひとのふり見て我ふり直せ」の諺が，端的にこれを物語っていると言えよう．しかし，ユーザーの好みに応じて造成された仮想環境では，人間の長い社会生活の営みの中で自然発生的に生まれた，この教育システムというものが提供されない．この問題を解決するひとつの方法として，社会性を何らかの形で仮想環境の中にも含有した仮想空間の実現が必要であると考えられる．

我々人間が，最も日常的に使用する社会的コミュニケーション手段は，ことばである．しかし，ことばは，1対1でのコミュニケーション，あるいは，ひとりから多数に対するコミュニケーションにおいて，発話者ひとりの考えや雰囲気を他者へ伝達するものである．したがって，ことばというコミュニケーション手段によって「場」の雰囲気を生成するためには，ある程度の時間をかけて会話をしなければならない．これとは逆に，大勢の意見をひとりに短時間のうちに伝えるために我々が日常的に用いる聴覚的コミュニケーション手段は拍手である．そこで，本研究では，聴覚刺激による身近な社会的コミュニケーション手段である拍手に着目する．人は，大勢の拍手音から，その空間の広がりや「場」の雰囲気を感じ取ることができる．したがって，本研究の立場は，実際に起こり得る自然の拍手音と区別が付かない拍手音を自在に合成して提示することができれば，仮想環境における「場」の雰囲気の制御の助けになるのではないかというものである．例えば，定評のある外国の有名な音楽ホールを仮想環境で再現し，3次元立体音像技術を駆使して音響的にも臨場感を高めることができたとする．演奏が終わり，ユーザーが思わず拍手をしたとしても，周囲から沸き起こる予め録音されていた拍手とタイミングが異なっていたり，拍手の仕方が違っているのでは，その場から自分が遊離した感覚を受けてしまい，没入感を著しく低下させてしまう恐れがある．しかし，もし，自分が拍手を始めた時に，それに続いて周囲から同様の拍手が沸き起きるのなら，周囲の他の聴衆との間に一体感が生まれ，没入感を増大させることができるものと期待される．また，ユーザーの仮想環境内での行動や判断が正しいのかどうかを，適当な場面でユーザーに非言語的手段を用いてフィードバックすることもできることになる．

2　拍手に関するこれまでの研究

拍手は古今東西を問わず，人間により行われてきた社会的行動のひとつであり，その役割も文化圏により様々である．拍手を称賛の意を表現する手段として利用したのは，主に西洋である．このようなことから，拍手は古くから社会学者や人類文化学者などの興味を引いていた．しかし，優先度の低さゆえに，これまで，あまり多くのことが調べられていないのが現状である．

音響工学的な立場からは，拍手の音の種類をその振幅スペクトルの観点から解析を行った研究が挙げられる．Repp は，様々な人に拍手をさせ，その振幅スペクトルに対して主成分分析を行った．その結果，拍手音の代表的な 4 つのスペクトル形状を抽出し，これらが両手の合わせ方に依存して変化すると報告している [1]．また，久野は，どのような場面で手を叩くのかについてアンケート調査を行った．調査結果によると，拍手を行う理由は，喜び・感動 (44.0 %)，称賛・励まし (35.4 %)，形式的なもの (9.1%) などとなっている [2]．このことから，拍手には人の気持や想いが込められており，他人が行っている拍手音を聞くことにより，その人の想いを汲み取ることができるものと予想される．

拍手音の印象を左右する要素として，次の 5 つが考えられる．

1. 拍手音の大きさ
2. 拍手の間隔 (速さ)
3. 音色
4. 継続時間
5. ゆらぎ

さらに，周囲にも拍手をする人が存在する場合には，これらの要素に加えて，

6. 大きさの相対的関係
7. 速さの相対的関係
8. 手を叩くタイミングの相対的関係

などの要素も考えられる．このように，ある拍手を聞いた時に，その中に込められている拍手を行っている者の想いを知るためには，数多くの要因を解析しなければならない．

ここでは，この中から，主に拍手の速さとゆらぎに着目し，これらについて検討を行うことにする．拍手の速さは，拍手音の中に込められた拍手者の想いとの関係を調べるためである．一方，拍手のゆらぎは，人工的に合成した拍手音を，人間が行っている自然な拍手音に近づけるための検討である．6.～ 8. に記した要素について検討するには，周囲の拍手音に関する詳細な情報を得る必要があるが，これには，集団の拍手音から集団を構成する各人の拍手音を分離抽出しなくてはならない．これは，非常に実現が困難な課題である．また，各拍手の絶対的な大きさは，拍手者と聴取者との距離や拍手者の腕力に依存するものであるため，1. も今回の検討対象から除外した．一方，音色についても，主に Repp により研究されているスペクトル形状に依存する問題であるため，ここではこれらの要因を取り除いて検討することにする．

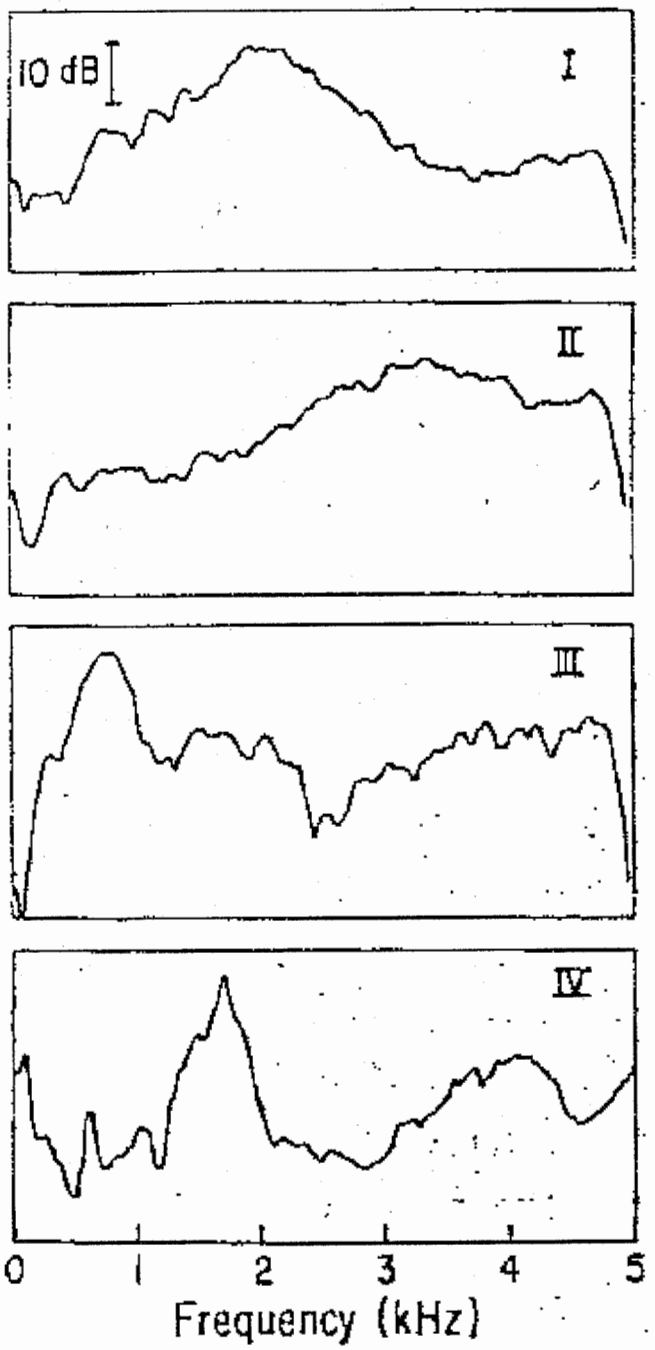


図 1: 代表的な 4 つの拍手の振幅スペクトル形状 (B. H. Repp, 1987)

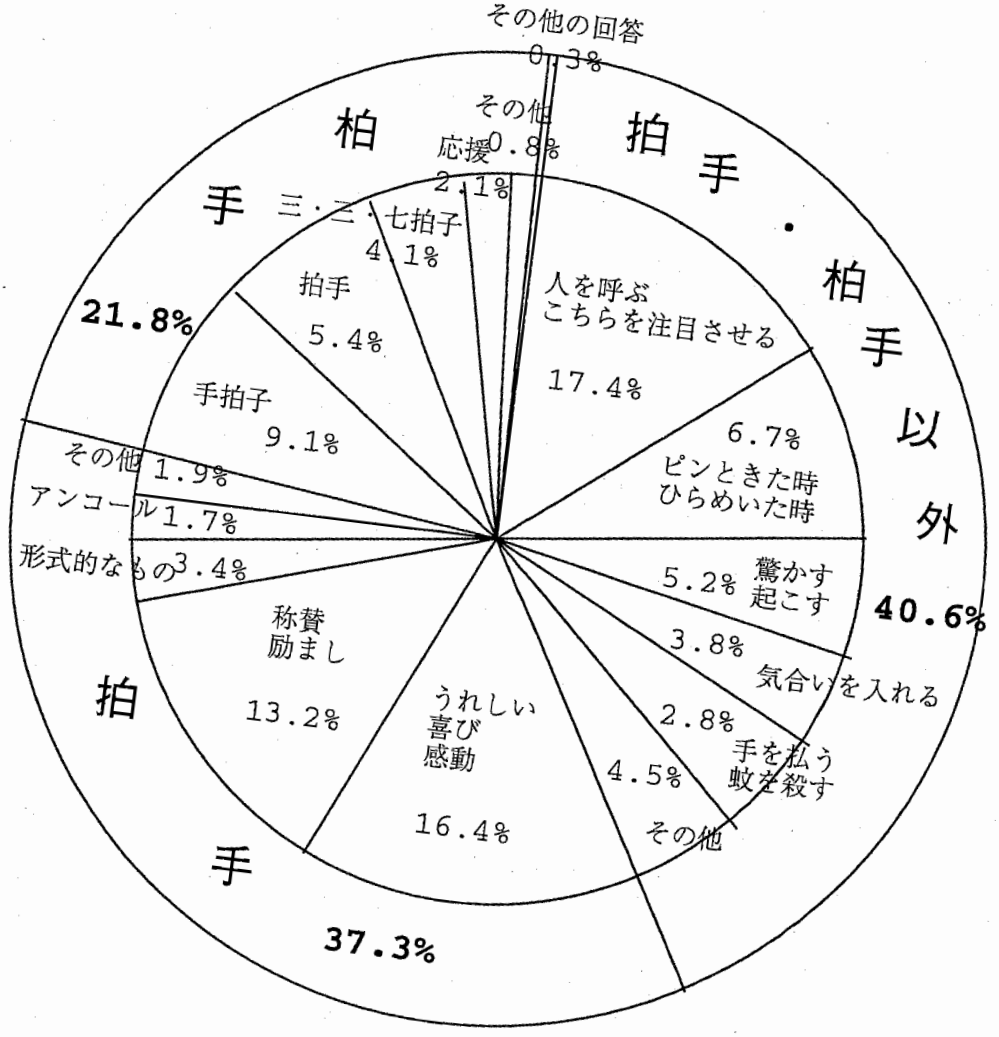


図 2: 人が手を叩く場面に関するアンケート結果 (久野, 1992)

表 1: 被験者に提示した拍手音刺激と諸特性

番号	音の種類	収録場所	継続時間	備考 (実験者の主観)
No.1	拍手のみ	会議室	15.6[s]	
No.2	拍手のみ	会議室	15.8[s]	人数多い
No.3	拍手のみ	会議室	17.3[s]	さらに多い，ピッチも速い
No.4	拍手のみ	ホール	6.1[s]	すぐに止む
No.5	拍手のみ	ホール	16.4[s]	やや音が高い，閑散とした感あり
No.6	拍手のみ	ホール	17.6[s]	No.5 より人が多い感じ
No.7	拍手のみ	ホール	17.7[s]	響きの音が高い，ピッチが高い
No.8	ざわめきあり	ホール	7.6[s]	継続時間が短い
No.9	声あり	屋外	15.9[s]	継続時間が長い
No.10	声あり	ホール	16.8[s]	No.9 よりも声のトーンがやや低い

3 実拍手音から状況を推測する能力

3.1 実験の目的

ある拍手音を聞いたときに，人はその音から「場」の雰囲気を読み取ることができる．しかし，ひとつの拍手音から読み取られた「場」の雰囲気が，多くの人達の間で共通でなければ，拍手を「場」の雰囲気の制御に利用することはできない．そこで，人が同じ拍手音を聞いたときに，どのような印象を感じ取っているのかを調べる必要がある．このことを調査するために，次に説明する実験を行った．

実験には，様々な状況下で予め録音された拍手音を提示する．実験に使用したのは，効果音用コンパクトディスクに納められている拍手音であり，サンプリング周波数が 44.1 kHz，量子化ビット数が 16 ビットのステレオ音源である．この中から，実験には，表 1 に示した 10 パターンの拍手音を使用した．これらは，実験者が収録されているすべての拍手用効果音を実際に聴取した上で，その中から異なる印象の刺激音が含まれるように取捨選択したものである．拍手者の人数，拍手のピッチ，部屋空間の違い，歓声の有無などの影響を比べられる刺激群となるように留意して選択を行った．No.1 から No.3 の刺激音群，あるいは，No.4 から No.7 の刺激音群を比較することにより，拍手者の人数が変化した場合の印象の違いを調べることができる．また，No.1 から No.3 の刺激音群と No.4 から No.7 の刺激音群とを比較することにより，部屋の違いに起因する印象の変化の傾向を知ることができる．No.8 から No.10 の刺激音群は，拍手音に加えて歓声音が含まれている場合の効果を調べるためのものである．

3.2 評価語による評定尺度実験

表 1 に記した拍手音を被験者の両耳にヘッドホンにより提示し，「称賛の意」，「盛り上がり」のふたつの評定語に対して，

1. 全くない

2. 少しある
3. 普通ぐらい
4. かなりある
5. 非常にある

の5段階で評価させた．10パターンの刺激音をランダムな順番で提示し，被験者は，各刺激音が提示される毎に評価を行った．以上を1回のセッションとして，1回のセッションにつき，ひとつの評価語に対して評価を行わせた．したがって，被験者は「称賛の意」と「盛り上がり」のそれぞれに対して，1セッションずつ回答を行うことになる．被験者の総数は，12人である．

「称賛の意」と「盛り上がり」に対する判断結果を，それぞれ，図3(a)と図3(b)に示す．これらの図において，円の直径が，各評価値を選択した被験者の数に対応している．No.1 から No.3 は，同じ会議室で録音されたものであるが，図3(a)を見ると，刺激音番号が大きくなるに従って，「称賛の意」の評価値が高くなっている．また，No.4 から No.7 は，同じホールで録音されたものと判断されるが，刺激音番号が大きくなるに従って，こちらも評価値が高くなっていることが分かる．このことから，実験者が各刺激音を聞いて感じたのと同様に，被験者も刺激音番号が増えるにしたがって「場」の違いを感じ取り，それを「称賛の意」の増加として認識したものと推察される．また，この判断の傾向に関して，被験者間であまり大きな差は見られていない．したがって，拍手音を適切な制御の下に仮想環境利用者に提示することにより，その仮想環境の「場」の持つ雰囲気を利用者に提示できる可能性がある．

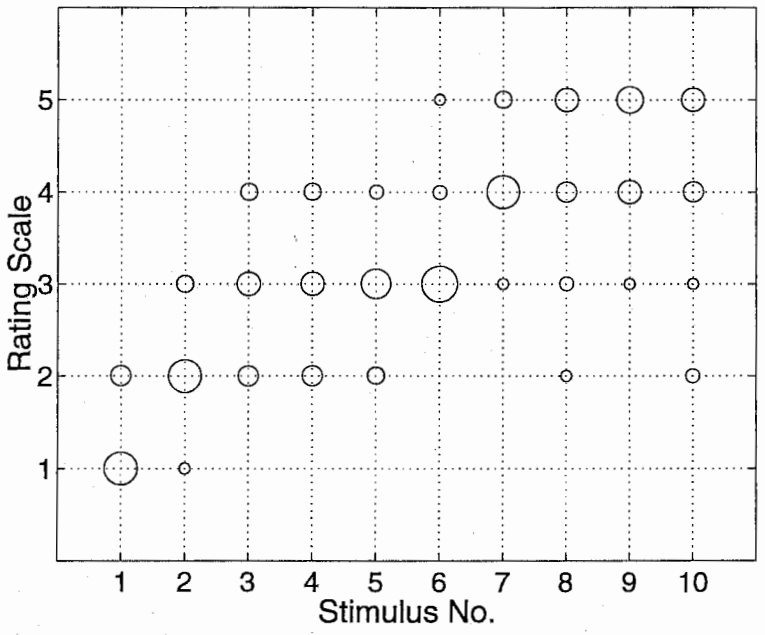
3.3 考察

図3(a)と図3(b)を見比べるかぎり，評価語が「称賛の意」の場合と，評価語が「盛り上がり」の場合とで，結果にあまり大きな差は見られない．ただし，図3(b)の No.8, No.9, No.10 において被験者が回答した評価値が高くなっていることから，歓声が入ることで「盛り上がり」感が顕著に増加するようである．図2に示した久野によるアンケート調査によると，「うれしさ」や「感動」を表現するためにも拍手という行動が取られるが，それらが集まっただけで，十分に「盛り上がり」を生み出せるというようなものではないようである．この事から，「盛り上がり」も制御するためには，歓声音の付加が有効なのかもしれない．歓声は，個々に発声している時点では言語的表現であるが，それが集まった音を聴取する側にとっては，むしろ，非言語的情報伝達手段としての色合いが濃くなってくる．このような意味において，歓声も拍手音と同様な効果を果たすのではないかと予想されるが，これについては今後の課題である．

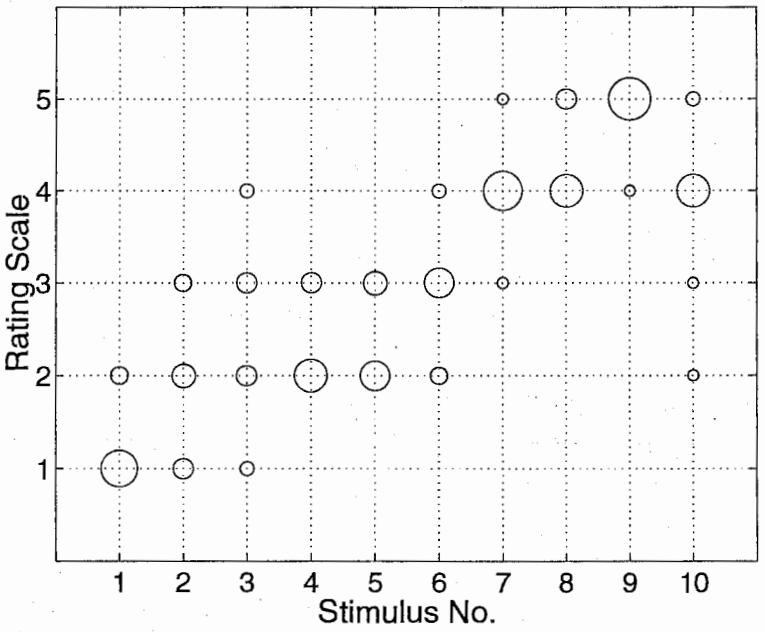
4 実拍手音からのパラメータ抽出

4.1 実験の目的

拍手において，手を叩く間隔や手を叩く強さは，必ずしも完全に一定ではない．むしろ，ゆらいでいるのが自然であると考えられる．実際，多くの生物リズムの変動が，1/f ゆらぎを呈することが調べられている．拍手の間隔のゆらぎについても，1/f ゆらぎを



(a) 評定語「称賛の意」に対する判断



(b) 評定語「盛り上がり」に対する判断

図 3: 評定尺度による群衆の拍手の印象評価

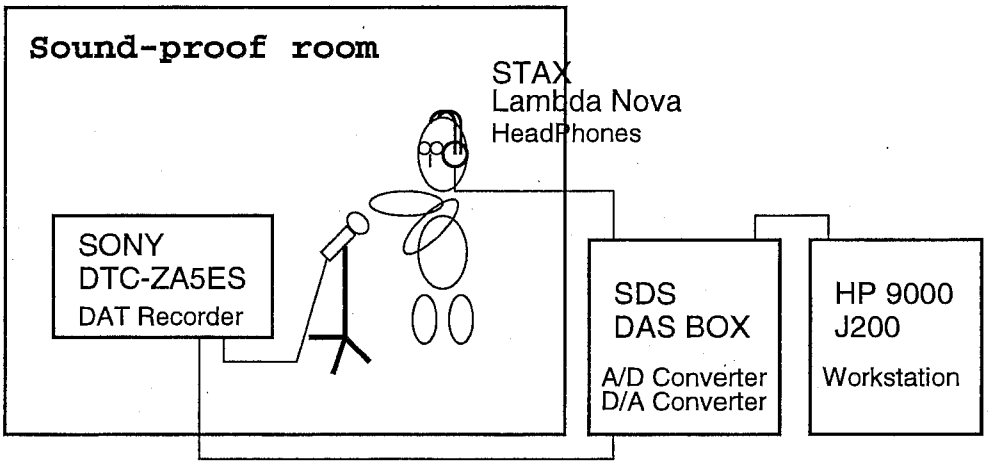


図 4: 拍手音測定のための実験系

有するという報告がなされている [3]. このことから，自然な印象を与える拍手を生成するには，適当なゆらぎを与える必要がある．ゆらぎの大きさは，拍手を行っている人間の気持ちによっても変化することが予想されるが，このような心理的要因による影響だけでなく，生理物理学的な要因による影響もあるものと考えられる．この後者の要因によるゆらぎは，対象としているパラメータの現在の値に依存して変化すると考えたほうが自然である．したがって，その依存関係を明らかにしておけば，合成拍手音をより自然な感じに近づけるための有用な知見となる．そこで，ここでは，拍手における手を叩く間隔とそのゆらぎの関係について，検討を行うことにする．

4.2 拍手の速度と拍手間隔のゆらぎ

被験者には，図 4 に示すように，ヘッドホンを通して実空間で録音された拍手音を提示し，「その場に居るつもりで拍手を行ってください」という教示の下で，防音室内でマイクロホンに向かって拍手を行わせた．こうすることにより，被験者の拍手音のみのデータを採取することができる．録音した拍手音に対して，ひとつの提示刺激音に対する拍手の間隔 (本報告では，これ以降，これをピッチと呼ぶ) の平均を横軸とし，その標準偏差を縦軸として図示したものが図 5 である．図 5 は，縦軸が対数スケールで表示されている．ピッチは，各拍手の音圧の瞬時値の絶対値が最大となるポイント間の長さから求めた．拍手音のサンプリング周波数は，48 kHz である．

図 5 から，相関係数として 0.68 という値が得られた．したがって，ピッチのゆらぎの量 (標準偏差) は現在のピッチからおおよその値が推定できることが期待される．そこで，実験データを回帰直線に帰着させることができるかどうかを調べるために，回帰性の有無の検定を行った．表 2 に示す分散分析表の結果から，$f \leq F^1_{148}(0.01)$ となり，ピッチとそのゆらぎの間には，回帰性があることが確認された．そこで，次に図 5 のデータを基に回帰直線を求める．拍手の平均間隔を x，その時のゆらぎの量の対数を y とすると，

$$y = 0.2908 + 0.0031x \tag{1}$$

という関係式が導かれる．図 5 は，縦軸を 10 を底とする対数で表した片対数グラフであ

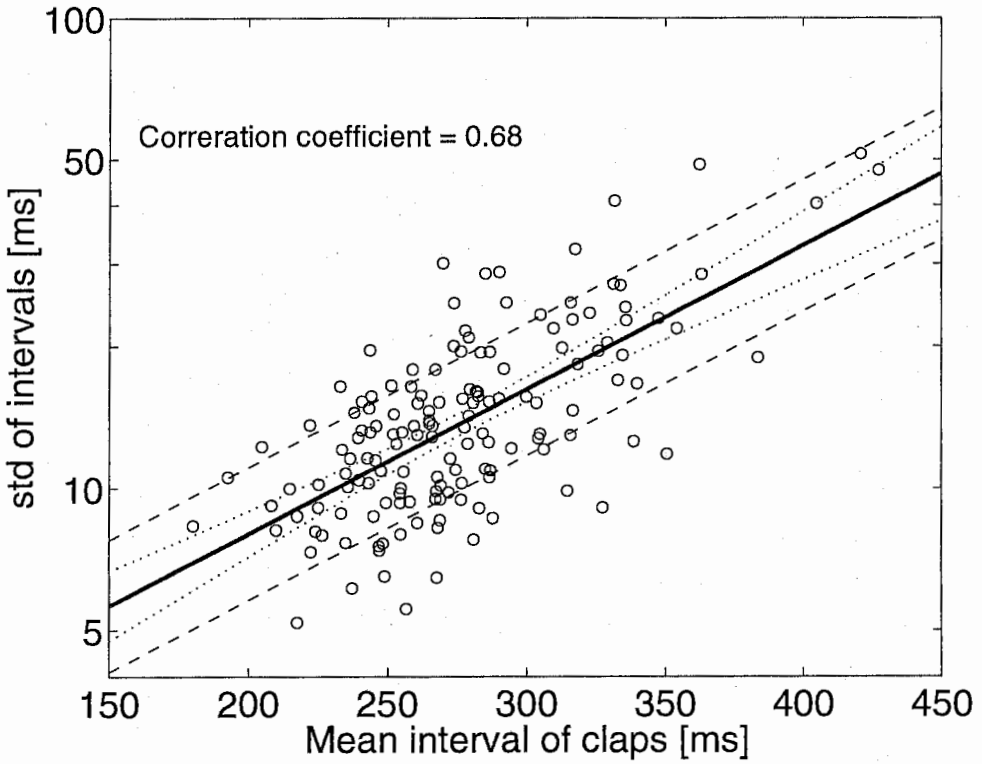


図 5: 拍手間隔の平均値と個人内分散との関係 (実線：回帰直線，点線：回帰直線の 5%信頼区間，鎖線：標準誤差)

表 2: 回帰性の有無の検定のための分散分析表

要因	平方和	自由度	不偏分散	不偏分散比
回帰変動	2.49	1	2.49	f=126.16
残差	2.92	148	0.02	
全変動	5.42	149		

るので，拍手のピッチを x とした時のゆらぎを与える式として，

$$\begin{aligned} \mathrm{std} &= 10^{0.2908+0.0031x} \\ &= 1.95 \times 10^{0.0031x} \end{aligned} \tag{2}$$

が得られる．

次に，各刺激音に対する拍手のピッチの変化について見てみる．図 6(a) は，各刺激音が提示された時の，拍手間隔を被験者毎に平均し，同一被験者の結果を直線で結んだものである．刺激音番号に系統的な意味合いは持たせていないため，本来，これらは直線で結べるものではない．しかし，同一被験者の刺激音の変化に対する拍手のピッチの変化を見るために，ここでは，便宜上，直線で結ぶことにした．

図 6(a) を見ると，被験者によって同じ刺激音を聞いた時に行う拍手のピッチが大きく異なっていることが分かる．しかし，各被験者毎に見ると，刺激音の違いによる拍手の速度の変化は必ずしも大きなものではない．ピッチが低い被験者は，刺激音の種類に関係なく，一様にピッチが低くなっている．一方，図 6(b) は，図 6(a) に示したデータに対して各刺激音ごとに平均と標準偏差を求めて図示したものである．これを見ると，同じ空間では刺激音番号が大きくなるに従って，被験者が行う拍手のピッチも速くなっていることが分かる．したがって，前節の評定尺度実験の結果と合わせて考えると，自由に速度を制御できる拍手音を合成し，そのピッチに応じたゆらぎを与えて，人工的ではなく自然な感じを持たせた上で仮想環境のユーザーに提示することにより，仮想環境の「場」の雰囲気を制御することができるものと期待される．

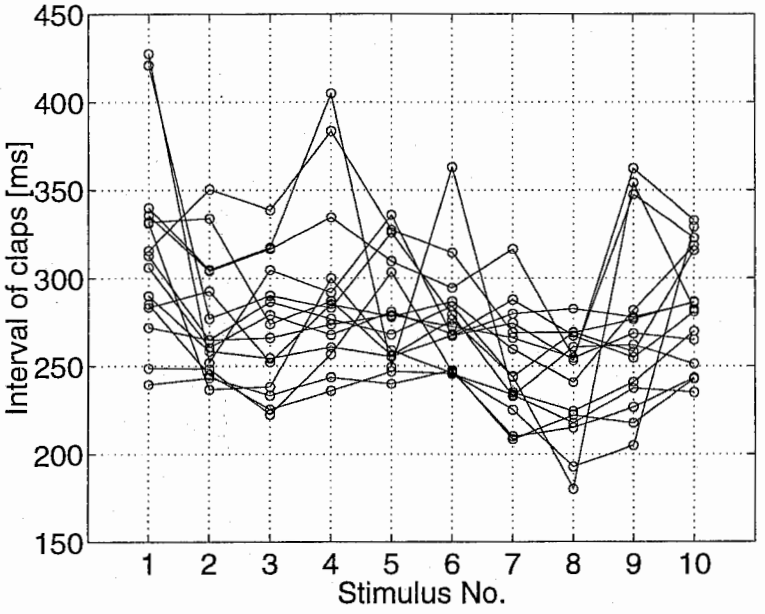
4.3 各拍手のパワーのゆらぎ

感動や賛美の程度によって，手を叩く強さが変わることは，日常的な経験から明らかである．これに伴って，拍手音の音色も変化することになる．しかし，音の強さは，各人の腕力に依っても変わるものである．また，拍手音を発している人と，それを聞いている人との距離にも強く依存する問題である．そこで，ここでは強さの絶対値ではなく，ひとりの人が同じ状況下で拍手をする中で，手を叩く度に拍手の音響的なパワーが，どの程度ゆらいでいるのかを調べることにする．

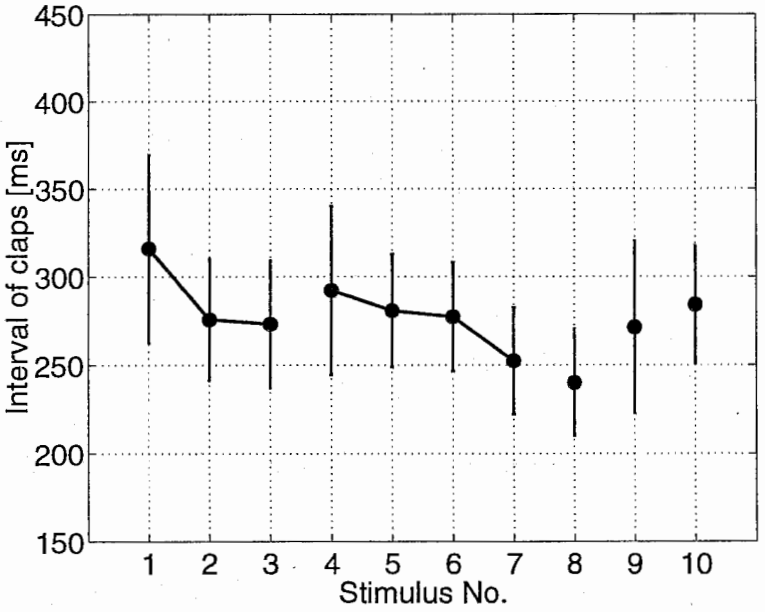
これに先立って，拍手音の音響的パワーの変動率 ρ を，

$$\rho = \frac{\sqrt{\frac{1}{L}\sum |P_i - \overline{P_i}|^2}}{\overline{P_i}} \tag{3}$$

により定義する．ここで，P_i は i 番目に手を叩いた時の音響的パワーを意味し，その平均値を $\overline{P_i}$ で表記している．式 (3) の物理的意味は，音響パワーの標準偏差を平均パワーで正規化したものと解釈することができる．前節の実験で得られた拍手音を基に，音響的パワーに関して解析を行った結果を図 7(a)~7(c) に示す．図 7(a) は，各刺激音に対する拍手の音響的パワーの変動率をプロットして，被験者毎にそれを直線で結んだものである．図 7(b) は，被験者間で平均を取り，標準偏差を誤差棒として示したものである．また，図 7(c) は，各被験者の拍手の音響的パワーの変動率と刺激音間での変動の関係を示したものである．前節では，拍手の速度とそのゆらぎの間に一定の関係があることが確認されたが，図 7(c) から，各拍手の音響的パワーに関しても同様の関係が成立するこ

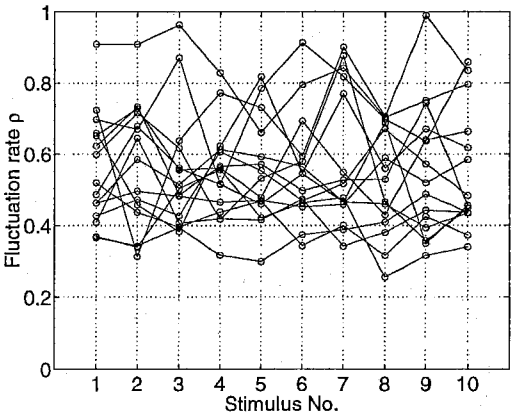


(a) 刺激音が提示された時の各被験者の拍手間隔

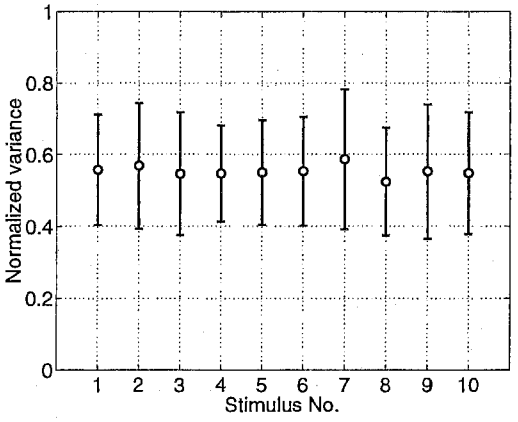


(b) 刺激音が提示された時の拍手間隔の被験者間にわたる平均と標準偏差

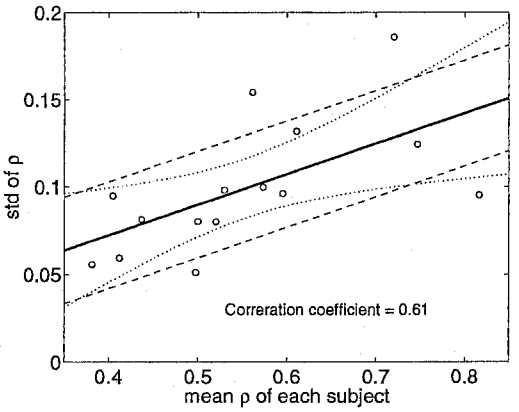
図 6: 各拍手音刺激を提示されたときの拍手の速度



(a) 各刺激音に対する拍手パワーの変動率



(b) 拍手パワーの被験者間の平均と標準偏差



(c) 各被験者の拍手パワーの変動率と刺激音間での変動の関係

図 7: 拍手パワーの変動率とそのゆらぎの提示刺激の違いによる変化

表 3: 回帰性の有無の検定のための分散分析表

要因	平方和	自由度	不偏分散	不偏分散比
回帰変動	0.0071	1	0.0071	f=7.66
残差	0.0120	13	0.000923	
全変動	0.0191	14		

とが予想される．そこで，パワーの変動率とそのゆらぎの間にも回帰性があるのかを検定した．表 3 に示した分散分析表から $f \leq F_{13}^{1}(0.01)$ となり，回帰性があることが判明した．そこで，図 7(c) から回帰直線を求めると，

$$y = 0.0027 + 0.1742x \tag{4}$$

となる．以上の結果を，ピッチの変動に関する式 (2) と合わせて，拍手音合成の際のパラメータとして利用することにより，合成的ではない自然な印象を与える拍手音の合成が可能になるものと予想される．

5 導出パラメータの検証実験

前節において，ピッチとゆらぎの関係，および，拍手の音響的パワーのゆらぎに関する関係式を導出した．そこで，この式 (2)，(4) を使用して合成された拍手音が，単純な単発拍手音の繰り返しにより合成された拍手音と比較して，自然な感じが増加しているのかを調べるために，簡単な検証実験を行った．

実験は，

1. 実際の拍手を録音した音 (8)
2. ピッチ・振幅ともにゆらぎを与えて合成した音 (7)
3. ピッチのみにゆらぎを与えて合成した音 (4)
4. 振幅のみにゆらぎを与えて合成した音 (10)
5. ピッチ・振幅ともに一定として合成した音 (1)

の 5 通りの拍手を聞かせて，被験者にはどの音がどのような処理をしたものなのかを告げずに，それぞれの拍手が「自然」か「不自然」かを答えるというものである．5 人の被験者に対し，各条件が 2 回ずつ含むようにして，ランダムな順番で刺激音を提示して判断させた．各条件の右側の括弧の中の数字は，実験の結果得られた，各拍手に対して自然だと答えた割合である．振幅のみにゆらぎと与えるという条件が，最も高い割合で，「自然だ」という回答が得られている．ピッチにもゆらぎを与えた場合に「自然だ」と答えた割合は 7 割となっているが，実際の拍手音に対しても，「自然だ」という回答が得られたのは 8 割程度なので，本実験で得られた知見を利用して拍手音を合成することにより，実際の拍手に似通った拍手音が合成できているものと判断される．実際の拍手音を聞かせた時にも「不自然」という回答が得られているのは，

1. ひとりだけで拍手をしている音である

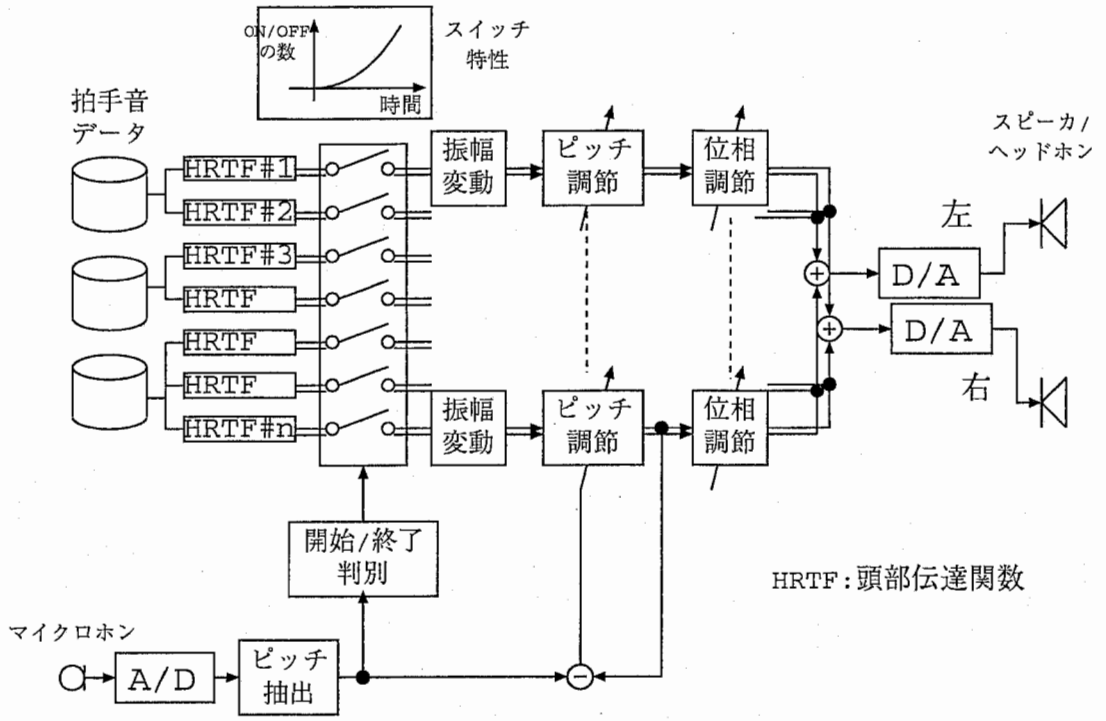


図 8: 実拍手誘導型自動拍手システムのブロックダイアグラム

2. 防音室という比較的反射音の少ない空間で録音されている

など，日常生活の中で拍手を聞く場面や空間と異なっていたためではないかと推察される．

6 実拍手誘導型自動拍手システム

6.1 システム概要

これまでの検討結果を基にして，あるユーザーが拍手を行ったときに，同じ想いを持った拍手音を合成してユーザーに提示するシステムを構築することができる．このシステムは，ユーザーが自分と共感を持つ人間の存在を，仮想空間の中に感じ取れることを目指したものである．ピッチが拍手者の想いに応じて変化することから，ユーザーと同じようなピッチを持つ複数の拍手音を合成することにより，この目的を実現しようという試みである．

ユーザーが拍手を始めると，仮想的な拍手者の数が徐々に増える．また，ユーザーが拍手を止めると，仮想的な拍手者の数も徐々に減り，最終的に全ての拍手が止むことになる．さらに，ユーザーが拍手の途中でピッチを変えると，仮想的な拍手者もそれに追従するようにピッチを変化させて行く．これにより，ユーザーは群衆の意思の先導者となり，自分の意志が大衆の支援を受けたと認識することにより，安心感や満足感を得ることができると期待される．そこで，前節までの検討で得られたパラメータを用い，図 8 に示すブロックダイアグラムにより，実時間で動作する実拍手誘導型自動拍手システ

ムを構築した．式 (2) および式 (4) は，図 8 における「振幅変動」および「ピッチ調節」というブロックでの信号処理に使用され，単発拍手音から自然な印象を与える拍手音を生成するために利用されている．

6.2 ハードウェア

構築した本システムのハードウェアは，

- ユーザーの拍手音を収音するための音響収集デバイス (マイクロホン)
- 生成した拍手音をユーザーへ提示するための音響提示デバイス (ヘッドホン，スピーカー)
- A/D，D/A 装置 (DASBOX，System Design Service 社)
- 信号処理計算機 (HP9000/700 J200)

により構成される．ここで，収音デバイスおよび音響提示デバイスに関しては種類を問わない．開発したソフトウェアは，A/D，D/A 装置として SDS 社製の DASBOX を使用することを前提とし，それを制御するための HP-UX 用ライブラリを使用している．DASBOX の詳しい仕様および使用方法については，DAXBOX 附属のマニュアルを参照して欲しい．信号処理計算機に関しての制約は特に無いが，DASBOX 制御用のライブラリがインストールされている必要があるため HP9000/700 J200 を使用している．

WS と独立した A/D，D/A 装置を使うのではなく，WS に内蔵されているサウンド入出力デバイスを使用することも可能である．SGI のマシンについては，IRIS マルチメディアライブラリに含まれるオーディオライブラリを用いて実現したプログラムもある．これについては，付録 D において説明する．

6.3 ソフトウェア

ここでは，HP9000/700 J200 を使用したプログラムについて説明する．実拍手誘導型自動拍手システムを 1 台の WS で実時間で動作させるにあたり，全体の処理を役割に応じて幾つかの部分に分割し，各処理を並列に実行させることによりこれを実現している．図 9 は，並列に実行される各プログラムのお互いの関係を図示したものである．

6.3.1 A/D

本システムで目指しているのは，ユーザーが拍手という表現形態を通して表現した感情に対して，同意 (場合によっては否定) の意を示す拍手音を合成することにより，ユーザーが共感の意をその拍手音から汲み取ることのできるシステムを構築することにある．したがって，合成された拍手音が，ユーザーの行った拍手という行為 (Action) に対する返答 (Reaction) であるとユーザーに知覚されなければならない．拍手というものは，1 対 1 での会話などと異なり，ある程度の人数がいる状況において，個人の意思を表現するためにその行為が開始される．この行為は，必ずしも返答を期待するものではないので，半ば一方的な意思伝達表現である．したがって，一般的に言って，自分の拍手と周

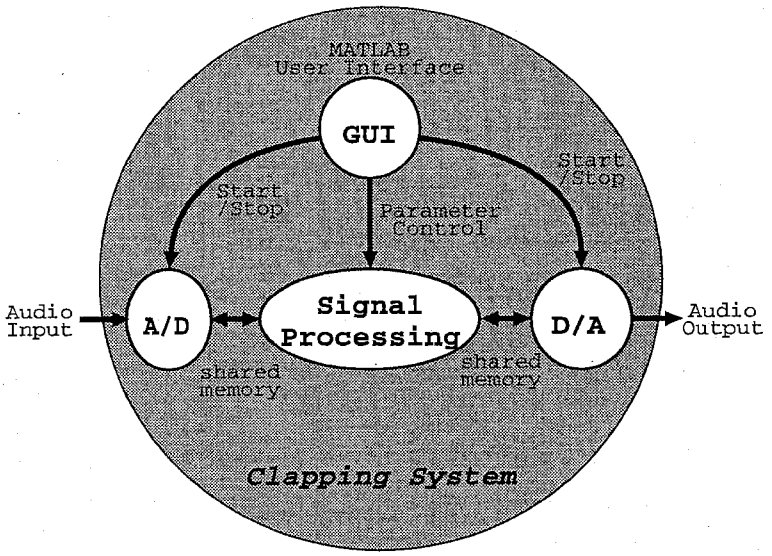


図 9: 本システムにおいて並列実行される各プログラムの関係図

囲から聞こえてくる拍手の間に，何らかの関係性を積極的には模索していない．そのような対象に対してもユーザーにインタラクション性を知覚させるには，何らかの意味において，ユーザーの行為とシステムの挙動に関連性を持たせなければならない．言葉のような論理的な通信手段と異なり，拍手は抽象的な表現形態による意思伝達手段であるため，第一段階において，意味的な繋がりから関連性を確立するのは容易ではない．そこで，時間的な繋がりから関連性を確立し，次段階の関連性の探索においてユーザーが意味的な関係を推測するという手順を取ることにする．換言すると，ユーザーの拍手が変化した時に，直ちに周囲の拍手も変化を生じさせることでユーザーに関係性を知覚させ，その後に，自分の拍手と周囲の拍手の相対的な違いから，周囲の人々の意思を推察させるということである．この場合，システムに要求されるのは，ユーザーの拍手音の変化に素早く反応できるということである．

拍手音はインパルス列に近い音響信号となるため，ユーザーが行っている拍手音を正確に解析して必要な情報を得るには，高いサンプリング周波数で A/D を行う必要がある．ところが，DASBOX は，FIFO 型のバッファを本体内に持ち，ホストワークステーションとの間を DMA 接続してデータの転送を行っている．ユーザーの拍手の変化に素早く反応したいという要求からは，DASBOX とホストワークステーションとの通信の回数を増やす必要があるが，これは通信のためのオーバヘッドを増加させてしまう．その結果，サンプリング周波数をあまり高くすることができない．一方，ユーザーの拍手音から取得したい情報は，拍手の間隔と強さである．インパルス的な信号となる拍手の強さを調べるためには，どうしても高いサンプリング周波数が要求されるが，上述の理由により，これは実現でない．そこで，A/D 部では，比較的低いサンプリング周波数でサンプリングを行い，できるだけ時間遅れを少なくして，拍手間隔の情報を取得する方法を検討することにする．

拍手の間隔を調べる方法としては，フーリエ変換を行って周波数領域で観測する方法や自己相関関数から求める方法などが考えられるが，信頼性の高い値を得るには，不確

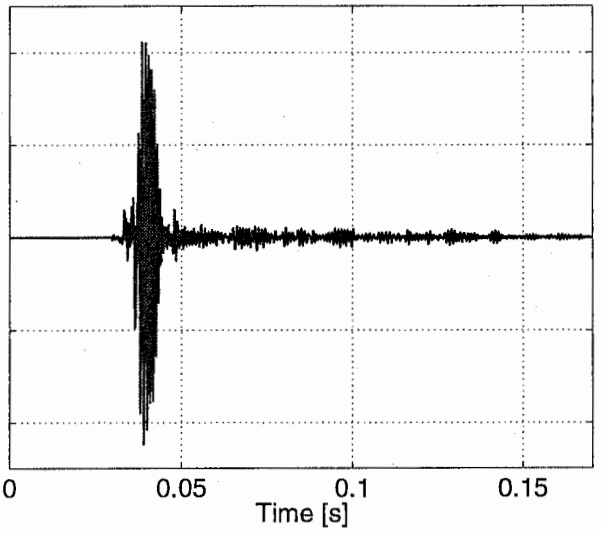


図 10: 1 拍の拍手音の時間波形

定性の関係から長い観測時間が必要となる．観測時間が長くなると，ユーザーの変化への追従までに時間遅れが生じ，その結果，ユーザーの拍手と周囲の拍手との関連性が薄弱になってしまう．この問題は，本システムの本来の目的の達成に対して，大きな障害となる恐れがあるため，これらの手法を採用することはできない．そこで，極めて単純であり，精度は必ずしも高くはないが，現時刻における拍手間隔を得られる方法を採用することにする．

拍手はインパルス的な信号であるが，図 10 に示すように，完全なインパルス信号とはならず，僅かではあるが時間的な広がりを持つ．そこで，サンプリング周期を，およそこの広がりの時間幅よりも若干小さな値に設定することにする．すると，サンプル値が大きな値を持つ場合には，その瞬間で拍手をしていると判断でき，逆に値が小さい場合には，その瞬間は拍手をしていないと判断することができる．また，1 拍の拍手の間にサンプリングが行われる回数も 1 回となる．したがって，サンプル値がある閾値よりも大きな値であったときに，以前に，同じように大きなサンプル値になった時刻から何サンプルが経過しているかを測定することにより，瞬時にその時刻における拍手間隔に関する情報を得ることができる．A/D 部を担当するプログラムでは，ここまでの処理を行い，得られた “現時刻における拍手間隔” を共有メモリへ書き出している．この A/D プログラムのフローチャートを図 11 に示す．

6.3.2 D/A

次節において説明する拍手音合成プログラムにより作成され，共有メモリに書き込まれている音信号データを D/A するのが本プログラムの役割である．ここで注意すべきは，DASBOX での A/D，D/A が，本体内の FIFO 型リングバッファを介して行われるのだが，このリングバッファに溜っているデータ量をモニターするライブラリが提供されていない点にある．したがって，リングバッファの残り容量に応じてホストワークス

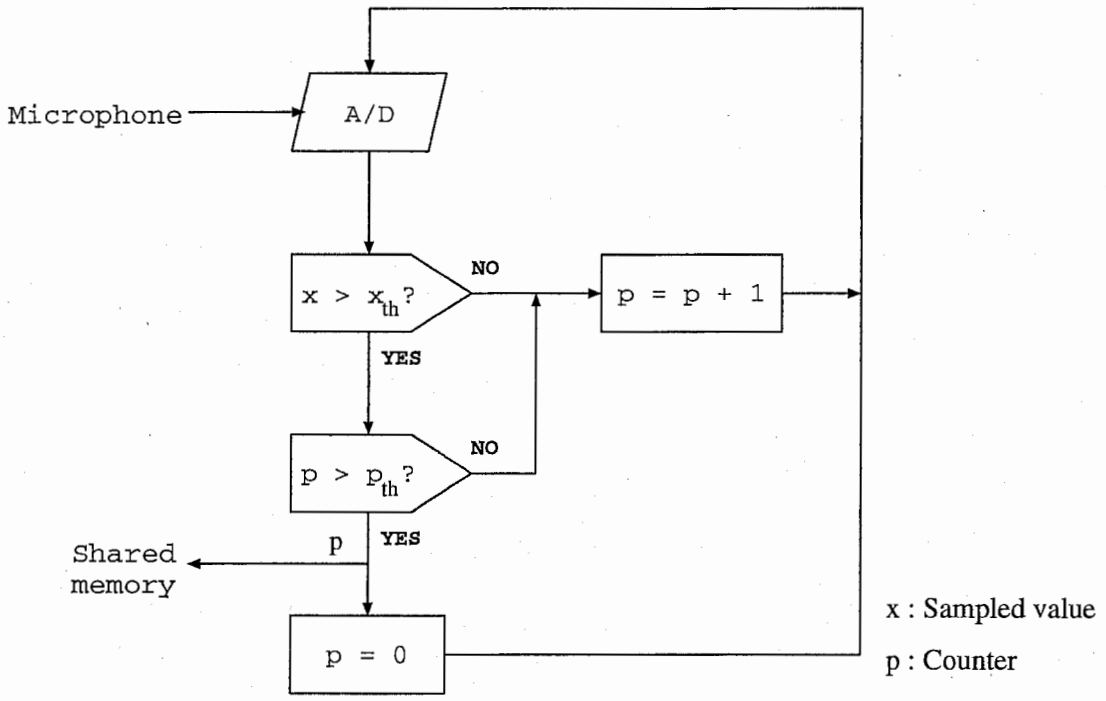


図 11: A/D プログラムでの処理のフローチャート

テーションから転送するデータの量を調節することができない．そこで，適当な時間長の一時停止を入れて，ホストワークステーションから DASBOX へデータを転送する時刻を調節する必要がある．もし，この時間が必要以上に長い場合には，DASBOX におけるリングバッファのデータが無くなってしまうためにエラーが発生し，DASBOX の D/A が停止してしまう．一方，この時間が短か過ぎる場合には，リングバッファに大くのデータが溜まるため，現時刻で再生すべき音信号データが，実際にはしばらく後になって再生されることになる．これは，A/D のところで問題として取り上げた，ユーザーの行為 (Action) とシステムの返答 (Reaction) の間の関連性を低下させる要因となるため，十分に注意しなければいけない問題である．加えて，この待ち時間の設定は，プログラムを実行するワークステーションの計算処理速度にも依存する問題である．なぜなら，DASBOX へデータを転送する前に，転送すべきデータをホストワークステーションが生成し終わっている必要があるからである．そのため，ホストワークステーションとして別のマシンを使用する場合には，改めてこの値を調整する必要がある．作成した D/A プログラムのフローチャートを，図 12 に示す．

6.3.3 拍手音合成プログラム

拍手音合成プログラムでは，ユーザーが実際に行った拍手の音から得られる情報を基にして，合成拍手音を生成する．本システムは，概要の節で述べたように，ユーザーの拍手がシステムの生成する拍手と何らかの関係性を持つものとして知覚されるようにするための配慮だけでなく，少しでも本物の拍手音に近いものを合成しつつも，使用者に制御性を持たせるための工夫が為されている．

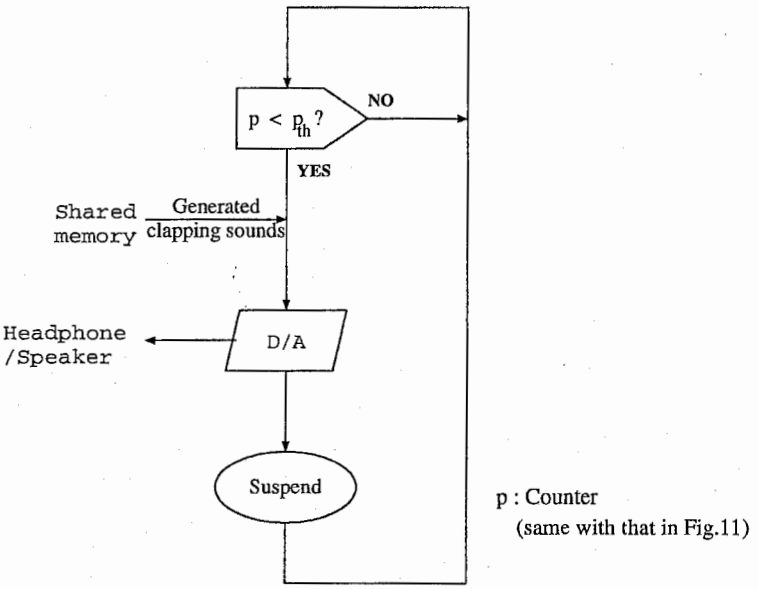


図 12: D/A プログラムのフローチャート

合成拍手音を本物と似せたものにするためには，以下に示す点に関して注意する必要がある．

1. 拍手音の自然性
2. 空間的な広がり
3. 拍手音 (者) の空間的な広がり方
4. 拍手音の同期性

以下では，それぞれの要素について，少し詳しく説明する．

拍手音の自然性 拍手の音色は，拍手をする際の手の重なり方によって，大まかに 4 通りに分類されるという報告がなされている [1]．したがって，各人各拍子の拍手音信号を完全に計算機で合成するのではなく，実際の拍手音をサンプリングしたものを記憶しておき，それを加工して利用するほうが，記憶容量や CPU パワーなどのコスト的負荷や，実時間で動作させるという必要要件，自然性の保持などの観点から適していると考えられる．そこで，本手法では，実際に人間が防音室内において行った拍手の音を録音し，その信号を各 1 拍子の拍手ごとに分離して，それを間隔や各拍子の大きさにゆらぎを与えながら提示することで，合成拍手を生成するという手法を採用している．自然な拍手の間隔や各拍子の大きさ，ゆらぎについては，先の検討により得られた値を使用した．こうすることにより，様々な拍手音をすべて記憶媒体に記憶して使用するのと比べ，大幅に必要とする記憶容量を削減することが可能となる．また，予め用意すべき拍手のサンプルも，数人の拍手音だけで十分になるという利点がある．

空間的な広がり 拍手音の録音は，拍手者とマイクロホンとの位置を固定して行っているため，その音をそのまま提示したのでは，全ての拍手が同じ位置から聞こえてきてしまう．このような状況は，現実には生じ得ないため，その音を聞いたとしても

現実味のない不自然な拍手音として知覚されてしまう．そこで，記憶されている各1拍子分の拍手に対して，空間的に異なる地点からの頭部伝達関数を畳み込むことにより空間的に広がりを持たせ，拍手音の空間的重なりを回避することにした．また，その際，各1拍の長さで記憶されている拍手に対して，それぞれ異なる頭部伝達関数を畳み込むことにより，各拍手音が空間的に広がってユーザーに聞こえるばかりでなく，拍手音の音色が変化することにより，数人の拍手音でしかなかった音が，異なる大勢の拍手音として知覚させることができるという利点もあげられる．

拍手音 (者) の空間的広がり方 大勢で拍手をする場合，何か極めて明確な合図となるものがあれば，全員が自律的に同時に拍手を開始することができる．明確な合図とは，例えば，演説者のお辞儀であるとか，舞台の幕が降り始めたなどのことである．しかし実際には，多くの場合において，誰か牽引役となる人が最初に拍手を始め，他の人はその拍手の音につられるようにして拍手を開始する．したがって，拍手音 (者) の広がりは，必ずしも急激に広がるものではなく，多くの場合，ある拍手をきっかけとして徐々にかつ確実に広がるものである．この現象を本システムでも再現するために，各仮想的な拍手者に対し，拍手を開始あるいは終了するかどうかを判断する仮想的な心理的閾値を設け，この閾値を徐々に変化されることにより，これを再現することを試みている．各仮想拍手者は，ユーザーが手を叩く毎に各々乱数を発生し，この乱数を閾値と比較して拍手の開始/終了の判断を行う．この遷移部での振る舞いは，プログラム中の該当する関数を適当に変更することにより，様々なものを実現することができる．図13に，最大拍手者数を50人に設定したときの，人数の増加の様子をシミュレーションした結果 (4回) を図示する．ここで使用した関数は，現時刻において拍手を行っている人数の最大拍手者数に対する比を x として，

$$f(x) = 1 - x^4 \tag{5}$$

を閾値とし，$0 \leq r < 1$ の範囲で発生させた一様乱数 r が $f(x)$ よりも大きい場合は，拍手を開始するというものである．したがって，拍手者が増加するに従って閾値が低下し，まだ拍手を行っていない者の拍手を容易に誘発させるという振る舞いをする．実際，この図から分かるように，拍手を行っている人数がある程度増えると，急激に最大値へ向かって増加している．また，拍手が増加し始めるまでの時間は試行ごとに異なり，振る舞いが単調化するのを防いでいるのがうかがえる．

拍手音の同期性 多くの人数で拍手を行う場合，人々がばらばらのタイミングで手を叩いていると，それは拍手になる．しかし，全員が同じタイミングで手を叩くと手拍子となり，それは拍手とは異なる意味を持った信号として人間には知覚される．したがって，大衆によって生起される拍手をシミュレートする場合においても，これらふたつのものを，全く独立なものとして生成するのではなく，類似した行為の中で生じ得るふたつの極端な状態としてとらえたほうが自然である．拍手から手拍子に連続的に変化されるには，ふたつのパラメータを適切に変化させればよい．それは，拍手における手を叩く間隔と，正に手を叩くタイミングである．拍手の間隔だけが同じで，手を叩くタイミングが合っていないと，いつまでも揃わない手拍子のようなものになってしまう．また，手を叩くタイミングが合っていても，拍手の

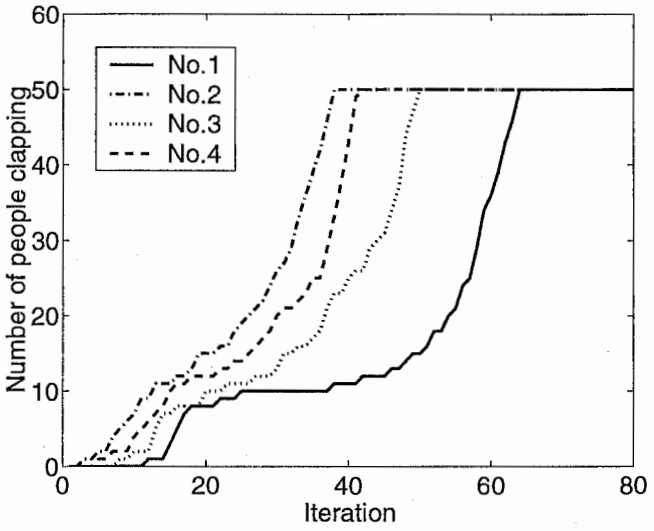


図 13: 仮想的拍手者の増加パターンのシミュレーション

間隔が一致していないと，手拍子の音の大きさが，叩く度に大きく変化してしまい，ひとつのまとまった意思としては知覚されず，幾つもの想いが交錯した手拍子として知覚されてしまう．そこで，本システムでは，これらふたつのパラメータを滑らかに変化させるために，適応アルゴリズムでしばしば用いられる，以下の更新式 (6) を利用して変化させている．

$$x(n+1) = x(n) + \mu\{x(n) - x_o\} \tag{6}$$

ここで，$x(n)$ は時刻 n におけるパラメータの値，$x(n+1)$ は時刻 $n+1$ におけるパラメータの値，x_o はこのパラメータの目標値を表している．本システムの場合，x は，ユーザーの拍手の間隔や手を叩くタイミングである．ここで，μ の値を小さく設定すると，仮想拍手者がそれぞれ初期値で設定されたパラメータにしたがって拍手を行うので，いつまでも手拍子にはならない．一方，μ の値をある程度大きな値に設定すると，拍手の時間間隔や，手を叩くタイミングの差は徐々に縮まり，拍手から手拍子へと次第に変化してゆくことになる．

以上の点を考慮して作成したプログラムのフローチャートを図 14 に示す．

7 結論

仮想環境のユーザーと，仮想環境内に存在する人間との間での聴覚的コミュニケーション手段として，拍手という行為に着目した．ここでのコミュニケーションとは，ユーザーと仮想環境内の人間が正対して意志の伝達をするというものではなく，第 3 者に対してユーザーが意志を表現した時に，仮想環境内の仮想的な人間が同意あるいは否定的な意志を非言語的な手段で表現することにより，ユーザーとの間に共感や励ましといった意志の伝達が形成されるという意味での深層的なコミュニケーションである．これを実現するためには，

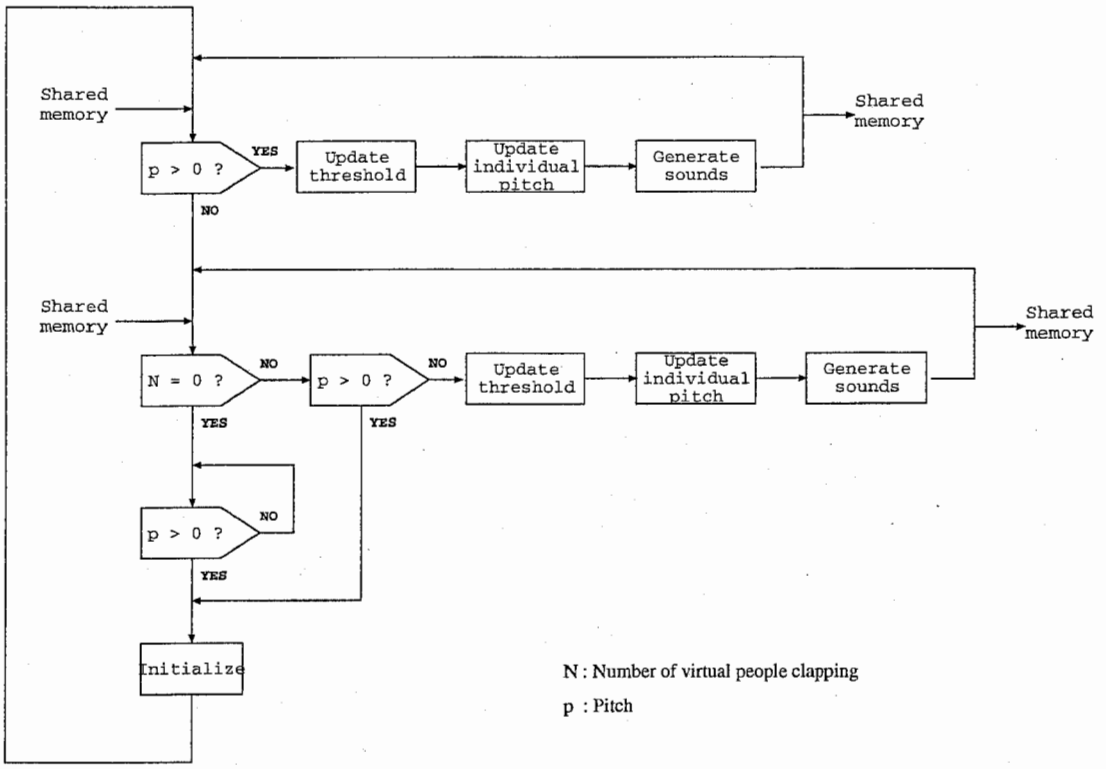


図 14: 合成拍手音の作成を行うプログラムもフローチャート

1. ユーザーの現在の意志を推定する
2. 仮想環境内の人間に，その意志と同様の意志を何らかの手段により表現させる
3. これらを実時間で実行する

という，3 つが実現されなければならない．

拍手では，これらの意志が，拍手の手を叩く間隔と関係して変化していることが実験から予想された．そこで，ユーザーの拍手からその手を叩く間隔を追跡し，それと同様の速度で手を叩いた時に生じる拍手音を合成してユーザーへ提示することで，上記の条件を満たすことを試みた．その際，合成する拍手音を人間が行った拍手音に似せる必要があるため，人間が拍手を行った場合の，手を叩く間隔のゆらぎ，および，振幅のゆらぎに着目してその関係を調べた．その結果，回帰式で比較的よく表現できることが判明したので，その関係式を用いて実時間で人間が行った拍手音と似せた拍手を合成することに成功した．そこで，この知見を用いて，1 台のワークステーションと A/D，D/A 変換器から成るシステムで，ユーザーがマイクロホンに向かって拍手を行うと，それに対して同感の意を現す拍手が聞こえてくるシステムを構築することに成功した．

本システムは，現存する装置を用いて実時間で動作させることを最重要視しているために，内部で行っている信号処理は極めて簡素なものである．実世界で生じる群衆の拍手には，個人の癖や集団心理，耳から聞こえてくる他人の拍手音によるフィードバックなど，まだまだ考慮しなければいけない要因がたくさん潜んでいる．したがって，拍手という一見単純な行為を取ってみても，非言語的手段による聴覚コミュニケーションにより意志の伝達を図るためには，社会心理学における知見を蓄えるのと同時に，高速な

計算アルゴリズムの開発などソフト・ハード両面において，今後更なる発展が必要となるであろう．

参考文献

[1] Bruno H. Repp, "The sound of two hands clapping: An exploratory study," *J. Acoust. Soc. Am.*, Vol. 81, No. 4, pp. 1100–1109, April 1987.

[2] 久野和宏, "拍手 (柏手) のこと," 信学技報 (EA-92 8-16), Vol. 92, No. 71, pp. 53–60, May 1992.

[3] Musha T, "Fluctuations of biological rhythm," *Fornt Med Biol Eng*, Vol. 3, No. 2, pp. 131–133, 1991.

付録

A　A/D 用プログラム

```
/*
$Header: /home/ryou/works/APPLAUSE/Real/RCS/tempad.c,v 2.6 1998/11/01 11:01:43 ryou Exp
*/
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <values.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <sys/shm.h>
#include <unistd.h>
#include "dasioctl.h"      /* DAS Box 制御関数ライブラリのヘッダ */
#include "das90.h"         /* DAS Box 制御関数ライブラリのヘッダ */

/**************************************
DAS Box でデータを A/D する.
     cblk:   DAS Box 制御ブロック
     bufptr: A/D データを収納するポインタ
**************************************/
das_adread2( cblk,bufptr,dmacnt )
struct control_block *cblk ;
int *bufptr ;
unsigned int dmacnt ;
{
/* lseek( cblk->sfd,0L,0 ); */
 if( read( cblk->sfd,bufptr,dmacnt * 2 ) != dmacnt * 2 )   return( -1 ) ;
 return( 0 ) ;
}

/*********************************************
制御ブロックの生成と DAS Box のイニシャライズ
     cblock: DAS Box 制御ブロック
     select: 信号入力元 (D/A チャネル) 選択
*********************************************/
void setup_ad( struct control_block *cblock, char *select )
```

```
{
  struct dasbox_arg dasarg;
  static char *addevice_name="/dev/das1";

  dasarg.dasmode = ADNORM;          /* Normal A/D */
  dasarg.dmatime = 3;               /* 3 seconds until time out */
  dasarg.attn = 0;
  dasarg.gain = 0;                  /* gain */
  dasarg.clkmode = INTCLK;
  dasarg.clock = 48000.;            /* sampling frequency */
  dasarg.frame1 = 48 * 1000000;     /* data length */
  dasarg.mutelevel = 0.0001;
  dasarg.channels = 1;              /* number of channels */

/* DAS Box の A/D のチャネル番号 */
  if( !strcmp( select, "booth" ) )
    dasarg.chanum[0] = 1;               /* input channel number */
  else
    dasarg.chanum[0] = 2;               /* input channel number */

  if( das_open( cblock, &dasarg, addevice_name ) < 0 ) {
    fprintf( stderr, "das_open: failed to open DASBOX via %s.\n", addevice_name );
    exit( 0 );
  }
  if( das_packet( cblock, &dasarg ) < 0 ) {
    printf( "PACKET CMD error\n" );
    exit( 0 );
  }
}

/************************************
A/D のメインルーチン
    argv[1]: 入力元を設定 [booth|side]
************************************/
main( int argc, char **argv )
{
  int rtnvl, taplen, cnt, flg, *pitch, myshmid2;
  short *data, old=MAXSHORT;
  struct control_block dacblock, adcblock;
  long frame;
  FILE *fpp;
```

```
  taplen = 1;          /* 1ポイントずつ A/D する */
  cnt = flg = 0;

/****
     ピッチ情報のプログラム間でのやり取りのために,
     pitch を shared memory で確保
****/
  myshmid2 = shmget( IPC_PRIVATE, sizeof(int)*2, 0600 );
  pitch = (int*) shmat( myshmid2, (int*) 0, 0 );
  *pitch = 0;
  fpp = fopen( "shmaddress2", "w" );
  fprintf( fpp, "%d", myshmid2 );
  fclose( fpp );

  data = (short*) malloc ( sizeof(short) * taplen );
  if( data == NULL ) {
    printf ( "error: no enough memory\n" );
    exit( 0 );
  }
  setup_ad( &adcblock, argv[1] );

/* 無限ループ */
  while( 1 ) {

/* A/D する */
    if( das_adstart( &adcblock ) < 0 ) {
      fprintf( stderr, "das_adstart: failed.\n" );
      exit( 0 );
    }
    das_adread2( &adcblock, data, taplen );

/* 絶対値を取る */
    if( *data & 0x8000 )      *data ^= 0xffff;

    if( flg ) {
/***
    data が, ある閾値よりも大きな値になった2点間の間隔を以って,
    ピッチとする. cnt は, 不感地帯を設けるためのもの.
    マイクのゲインとの兼ね合いで, 適切に設定する.
***/
      if( *data & 0xff80 && cnt & 0xffc0 ) {
*pitch = cnt;
```

```
cnt = 0;
      }
/****
     以下，拍手の間隔が開き過ぎたときに，
     「拍手をやめた」と認識させるためのもの
      0xfe00: "cnt" が  512 より大きいとき
      0xfc00: "cnt" が 1024 より大きいとき
      0xf800: "cnt" が 2048 より大きいとき
***/
      else if( cnt & 0xfe00 ) {
        *pitch = 0;

flg = cnt = 0;
      } else
cnt++;
    } else {

/* 途中で拍手を再開した場合 */
      if( *data & 0xff80 ) flg = 1;
    }
  } /* while(1) の終了*/

  das_close( &adcblock );

/* shared memory と heap memory の解放 */
  shmdt( pitch );
  free( data );
  return( 0 );
}
```

B D/A 用プログラム

```
/*
  $Header: /home/ryou/works/APPLAUSE/Real/RCS/tempda.c,v 2.8 1998/11/10 10:10:11 ryou Ex
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <strings.h>
#include <values.h>
#include <time.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <sys/shm.h>
#include <unistd.h>
#include "dasioctl.h"
#include "das90.h"

static int sampling = 48000;            /* サンプリング周波数 */

/*************************************
DAS Box でデータを D/A する.
     cblk:   DAS Box 制御ブロック
     bufptr: A/D データを収納するポインタ
*************************************/
das_dawrite2( cblk,bufptr,dmacnt )
struct control_block *cblk ;
int *bufptr ;
unsigned int dmacnt ;
{
  int rval;
  if (( rval = write( cblk->sfd,bufptr,dmacnt * 2 )) != dmacnt * 2 ) {
    return( -1 ) ;
  }
  return( 0 ) ;
}

/*****************************************
制御ブロックの生成と DAS Box のイニシャライズ
```

```
      cblock: DAS Box 制御ブロック
      select: 信号出力先(D/A 出力チャネル)選択
*****************************************/
void setup_da( struct control_block *cblock, char* select )
{
  struct dasbox_arg dasarg;
  static char *dadevice_name="/dev/das0";
  char file[30];
  short *data;

/****
     DAX Box 用諸設定
****/
  dasarg.dasmode = DANORM;
  dasarg.dmatime = 1;
  dasarg.attn = 0;
  dasarg.gain = 0;
  dasarg.clkmode = INTCLK;
  dasarg.clock = (float) sampling;
  dasarg.frame1 = (int)pow( 2., 64.) - 1;
  dasarg.frame3 = 5;

/****
     D/A 出力チャネルの設定
     diotick  : 1 channel (1)
     speaker  : 2 channel (1,2)
     otherwise: 2 channel (2,3)
****/
  if( !strcmp( select, "diotick" ) ){
    dasarg.channels = 1;
    dasarg.chanum[0] = 4;
  } else if ( !strcmp( select, "speaker") ) {
    dasarg.channels = 2;
    dasarg.chanum[0] = 1;
    dasarg.chanum[1] = 2;
  } else {
    dasarg.channels = 2;
    dasarg.chanum[0] = 2;
    dasarg.chanum[1] = 3;
  }

/****
```

```
    DAS Box のオープン
****/
  if( das_open( cblock, &dasarg, dadevice_name ) ) {
    fprintf( stderr, "das_open: failed to open DASBOX.\n" );
    exit( 0 );
  }
  if( das_packet( cblock, &dasarg ) < 0 ) {
    printf( "PACKET CMD error\n" );
    exit( 0 );
  }
}

/************************
D/A のメインルーチン
    argv[1]: 出力先を設定
************************/
main( int argc, char** argv )
{
  int i, taplen, taplen2;
  int *pitch, *flg;
  int myshmid1, myshmid2;
  short *data, *shmdata;
  struct control_block dacblock, adcblock;
  long frame, cnt=-1;
  struct shmid_ds buf;
  FILE *fpp;

  taplen = sampling;
  taplen2 = sampling * 2;        /* stereo で再生する用 */

/****
     tempcc で作成されたデータは,
     shared memory を用いてこちらで吸い上げられる.
****/
  fpp = fopen( "shmaddress1", "r" );
  fscanf( fpp, "%d", &myshmid1 );
  fclose( fpp );
  shmdata = (short*) shmat( myshmid1, (short*) 0,  0 );

/****
     shared memory に格納されている pitch を読み出し,
     pitch==0 だと, 拍手が止んでいるものとする.
```

```
****/
  fpp = fopen( "shmaddress2", "r" );
  fscanf( fpp, "%d", &myshmid2 );
  fclose( fpp );
  pitch = (int*) shmat( myshmid2, (int*) 0, 0 );
  flg = pitch + 1;

  data = (short*) malloc( sizeof(short) * taplen2 );

/* 無限ループ */
  for(;;) {

    if( !*flg ) {

/* D/A のセットアップとスタート */
      setup_da( &dacblock, argv[1] );
      if( das_dastart( &dacblock ) < 0 ) {
printf( "das_dastart error\n" );
exit( 0 );
      }
#ifdef _DEBUG
      fpp = fopen( "temp.dat", "w" );
#endif

      while( 1 ) {
if( !*flg ) {
/****
     今現在の shared memory にある data を再生する
     (データを作っているのは, tempcc)
****/
  shmctl( myshmid1, SHM_LOCK, &buf );
  memcpy( data, shmdata, sizeof(short)*taplen2 );
  shmctl( myshmid1, SHM_UNLOCK, &buf );
  das_dawrite2( &dacblock, data, taplen2 );
#ifdef _DEBUG
  fwrite( data, sizeof(short), taplen2, fpp );
#endif
  *flg = 1;
  usleep( 800000 );  /* 処理時間調節 */
} else {
#ifdef _DEBUG
  fclose( fpp );
```

```
#endif
  break;
}
      }
/* D/A 用 DAS Box を閉じる */
      das_close( &dacblock );
    }
  }
/* for(;;) の終了 */

  shmdt( shmdata );
  shmdt( pitch );
  free( data );
  return( 0 );
}
```

C　拍手音合成用プログラム

```
/*
$Header: /home/ryou/works/APPLAUSE/Real/RCS/tempcc.c,v 3.8 1998/11/04 11:13:01 ryou Exp
*/

/*
  dataset.flg : 個々の拍手のステータス
        0x01 -> 拍手を行っている
        0x02 -> ユーザーの拍手が終了した
        0x04 -> 拍手が止んでいる.

  flg : tempda が新しいデータを受け付けられるかを示すフラグ
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <values.h>
#include <math.h>
#include <time.h>
#include <unistd.h>
#include <sys/shm.h>
#include "dasioctl.h"
#include "das90.h"

struct dataset {
  short *clap, flg;
  int pntr, pntnum, *pnt;
  double mu, nu, coeff, mypch;
};

/*******************************************
拍手音の種類を決める.
      _CLAP2048: 24 kHz サンプリングの拍手音
      _CLAP4096: 48 kHz サンプリングの拍手音
      _CLAP9192: 48 kHz サンプリングの足踏み音
*******************************************/
#define _CLAP4096

#ifdef _CLAP2048
#define CLAPLEN 2048
#define CLAPLEN2 4096
```

```
static char srcdir[] = "../clap24";
static char fhead[] = "clap";
static int shift = 5;
#endif

#ifdef _CLAP4096
#define CLAPLEN 4096
#define CLAPLEN2 8192
static char srcdir[] = "../clap48";
static char fhead[] = "clap";
static int shift = 6;
#endif

#ifdef _CLAP8192
#define CLAPLEN 8192
#define CLAPLEN2 16384
static char srcdir[] = "../bump48";
static char fhead[] = "dump";
static int shift = 6;
#endif

static struct dataset *data;
static int taplen;
static double thrshld1, thrshld2;
static double *prms;

/*******************************
  拍手音のデータを読み込んでくる
*******************************/
void load_data( short *idata, char* name )
{
  int flg;
  char buff[60];
  FILE *fpp;

  sprintf( buff, "%s/%s", srcdir, name );
  fpp = fopen( buff, "r" );
  fread( idata, sizeof(short), CLAPLEN2, fpp );
  fclose( fpp );
}

/*************************************
```

```
  各仮想拍手音のピッチをアップデートする
*****************************************/
int update_pitch( struct dataset *dd, int pitch )
{
  double mm;

  mm = prms[0] * dd->mu;
  if( mm >= 1. ) mm = 1.;
  dd->mypch = dd->mypch + mm * ((double)pitch - dd->mypch);
  return (int)dd->mypch;
}

/************
  各種初期化
************/
void init_param( int subj )
{
  int i;
  double ss, ee;
  char name[40];

  data = (struct dataset*) malloc( sizeof(struct dataset) * subj );

  for( i=0; i<subj; i++ ) {
/* 各単発拍手音を読み込む */
    data[i].clap = (short*) malloc( sizeof(short) * CLAPLEN2 );
    sprintf( name, "%s%02d.dat", fhead, i );
    load_data( data[i].clap, name );

/* データ初期化 */
    data[i].pnt = (int*) malloc( sizeof(int) * 20 );
    data[i].flg = 0;
/* ピッチのゆらぎの範囲を設定する (e.g., 0.8~1.5) */
    ss = 0.8; ee = 1.5;
    data[i].coeff = drand48() * (ee - ss) + ss;
    data[i].pntr = (int) floor((double) taplen * drand48());
/* ピッチの適応更新に使用されるステップパラメータ */
    ss = 0.5; ee = 1.5;
    data[i].mu = ss + (ee - ss) * drand48();
/* 位相の適応更新に使用されるステップパラメータ */
    ss = 0.0001; ee = 0.0004;
    data[i].nu = ss + (ee - ss) * drand48();
```

```
  }
  data[0].flg = 1;
}

/*************************************************************
  各単発拍手音を時間軸上のどの地点に布置するのかを決定する
*************************************************************/
void mkpoint( struct dataset *dd, int pitch )
{
  int cpt, num=0, *pntr;

  pntr = &dd->pntr;
/* サンプリング周波数調整 */
  pitch <<= shift;

  if( *pntr < taplen ) {
/* 拍手 ON/OFF テストを行い, ON の場合はポインタを打つ */
    cpt = *pntr;
    if( !dd->flg ) dd->flg = drand48() > thrshld1 ? 1 : 0;
    if( dd->flg==0x3 ) dd->flg = drand48() > thrshld2 ? 0x6 : 0x3;
    if( dd->flg&0x1 ) dd->pnt[num++] = cpt;
/* 1フレームの間, その処理を繰り返す */
    while( cpt < taplen ) {
      cpt += update_pitch( dd, (pitch > 0 ? pitch : (int)(dd->mypch*1.5)) );
      if( !dd->flg ) dd->flg = drand48() > thrshld1 ? 1 : 0;
      if( dd->flg==0x03 ) dd->flg = drand48() > thrshld2 ? 0x6 : 0x3;
      if( dd->flg&0x1 ) dd->pnt[num++] = cpt;
    }
    *pntr = cpt - taplen;
  } else
    *pntr -= taplen;

/* 拍手を行った回数をセットする */
  dd->pntnum = num - 1;
}

/***********************************
  各仮想拍手者の拍手音を足し合わせる
***********************************/
void mksignal( short* idata, struct dataset *dd )
{
  int i, j, num;
```

```
  short *buff, *d1;
  double amp;

  d1 = dd->clap;
  num = dd->pntnum;
  for( i=0; i<num; i++ ) {
/****
     振幅にゆらぎを与えながら，各拍手音を足し合わせる
     0.1 ≦ amp < 0.5
     (※ 正規分布が望ましいが，高速化のために一様分布)
****/
    amp = 0.1 + .4 * drand48();

    buff = idata + (dd->pnt[i] << 1);
    for( j=0; j<CLAPLEN2; j++ )
      *buff++ += amp * *d1++;
    d1 -= CLAPLEN2;
  }
}

/*****************************************
  基準となる拍手音に対する位相差を求める
*****************************************/
int diffe( double ptch, int rpnt, struct dataset *dd )
{
  int cpnt, num, cnt, rtn;
  int cand1, cand2;

  cpnt = cnt = 0;
  num = dd->pntnum;                    /* 仮想拍手者0番の拍手回数 */
  while( rpnt > cpnt && cnt < num )
    cpnt = dd->pnt[cnt++];

/* 拍手回数が同じ場合は，最後の拍手で比較 */
  if( cnt==num && cnt > 0)
    rtn = dd->pnt[cnt-1] - rpnt;
/* 拍手回数が違う場合は，最後のふたつの内，時間差が短いほうを採択 */
  else if( cnt > 1 ) {
    cand1 = dd->pnt[cnt-2];
    cand2 = dd->pnt[cnt-1];
    rtn = rpnt - cand1 > cand2 - rpnt ? cand2 - rpnt : cand1 - rpnt;
/* 拍手回数が1回の場合は，その拍手で比較 */
```

```
  } else if (cnt==1)
    rtn = dd->pnt[0] - rpnt;
  else
    rtn = 0;

/* 差がピッチの半分以下になったら，完全な同期に切り換える */
  rtn = ptch*0.5 > rtn ? rtn : 0;

  return rtn;
}

/*************************************************************
  ステップパラメータνに従って，適応的に位相差を0に近づける
*************************************************************/
void adjst1( struct dataset *dd, int subj )
{
  int i, j, dff;
  struct dataset *cdd;

  for( i=1; i<subj; i++ ) {
    if( dd->flg ) {
      cdd = &dd[i];
      for( j=0; j<cdd->pntnum; j++ ) {
dff = diffe( cdd->mypch, cdd->pnt[j], dd );
cdd->pnt[j] += (int) floor(cdd->nu * prms[1] * (double)dff);
      }
      cdd->pntr += dff;
      if( cdd->pntr < 0 ) cdd->pntr = 0;
    }
  }
}

/************************************
合成拍手音信号作成のメインルーチン
    (argv[1]: 仮想拍手者数)
************************************/
main( int argc, char** argv )
{
  int j;
  int i, *pitch, *flg, sflg=0, subj, dlen, sumflg, rstflg;
  int myshmid1, myshmid2, myshmid3, myshmid4;
  int size_s_C2, size_s_t2, size_s_d2, taplen2;
```

```
  double invsubj, *clpps, clpps2, clpps3;
  short *odata, *shmdata;
  time_t tloc=NULL;
  struct shmid_ds buf;
  FILE *fpp;

  taplen = 48000;        /* タップ長 (1秒相当) */
  taplen2 = taplen * 2;
  dlen = taplen + CLAPLEN;
  srand48(time(&tloc));

/****
     仮想拍手者の設定
     優先順位：引数 > ファイル > デフォルト値
****/
  if( argc == 2 )
    subj = atoi( argv[1] );
  else if( (fpp=fopen( "subjs", "r" )) ) {
    fscanf( fpp, "%d", &subj );
    fclose( fpp );
  } else
    subj = 10;

/****
     共有メモリのセットアップ
         myshmid1: 合成拍手音信号
 myshmid2: ピッチ情報
 myshmid3: μとνのステップパラメータ
****/
  myshmid1 = shmget( IPC_PRIVATE, sizeof(short)*taplen2, 0600 );
  fpp = fopen( "shmaddress1", "w" );
  fprintf( fpp, "%d", myshmid1);
  fclose( fpp );

  fpp = fopen( "shmaddress2", "r" );
  fscanf( fpp, "%d", &myshmid2 );
  fclose( fpp );
  pitch = (int*) shmat( myshmid2, (int*) 0, 0 );

  myshmid3 = shmget( IPC_PRIVATE, sizeof(double)*2, 0600 );
  fpp = fopen( "shmaddress3", "w" );
  fprintf( fpp, "%d", myshmid3);
```

```
  fclose( fpp );
  prms = (double*) shmat( myshmid3, (double*) 0, 0 );

  if( (fpp = fopen( "shm_client", "r" )) ) {
    fscanf( fpp, "%d", &myshmid4 );
    fclose( fpp );
    clpps = (double*) shmat( myshmid4, (int*) 0, 0 );
  } else {
    clpps = (double*) malloc( sizeof(double) );
  }

  shmdata = (short*) shmat( myshmid1, (short*) 0, 0 );
  odata = (short*) malloc( sizeof(short) * dlen * 2);
  bzero( shmdata, sizeof(short)*taplen2 );

/* パラメータ初期化 */
  flg = pitch + 1;
  *flg = 1;
  invsubj = 1./(double)subj;
  init_param( subj );

  size_s_C2 = sizeof(short) * CLAPLEN2;
  size_s_t2 = sizeof(short) * taplen * 2;
  size_s_d2 = sizeof(short) * dlen * 2;

#ifdef _DEBUG
  fpp = fopen( "temp.dat", "w" );
#endif

/* 無限ループ */
  while( 1 ) {

    if( *pitch && *flg ) {
      memcpy( odata, odata+taplen2, size_s_C2 );
      bzero( odata+CLAPLEN2, size_s_t2 );

/****
     過渡部の遷移パターンの制御
     拍手開始判定の閾値を徐々に下げ，拍手が起こりやすくする
****/
      for( i=0,sumflg=0; i<subj; i++ )    sumflg += data[i].flg;
      *clpps = (double)sumflg * invsubj;
```

```
        clpps2 = *clpps * *clpps * *clpps * *clpps;
        clpps3 = 1. - *clpps;
        thrshld1 = 0.99 - clpps2;
        thrshld2 = 0.95 - clpps3 * clpps3 * clpps3;
        for( i=0; i<subj; i++ )
mkpoint( &data[i], *pitch );

/****
     位相差を適応的に制御する
     (この行をコメントアウトすると間違いなく動作)
****/
        if( prms[1] > 0. )
adjst1( data, subj );

/* 拍手音を合成する */
        for( i=0; i<subj; i++ )
if( data[i].flg ) mksignal( odata, &data[i] );

/* 合成した拍手音信号を共有メモリへ書き出す */
        shmctl( myshmid1, SHM_LOCK, &buf );
        memcpy( shmdata, odata, size_s_t2 );
        shmctl( myshmid1, SHM_UNLOCK, &buf );

#ifdef _DEBUG
        if( j < 120 ) {
fwrite( odata, sizeof(double), CLAPLEN2, fpp );
j++;
        } else if( j==120 )
fclose( fpp );
#endif

        *flg = 0;
        sflg = 1;              /* 拍手は起こっているというフラグ */
    } /* while(1)の終了 */

/****
     ユーザーが拍手を止めた時の処理
****/
    if( *pitch==0 ) {
/* ユーザーが拍手を停止しているとうフラグをセットする */
        rstflg = 0;
        if( sflg )
```

```
for( i=1; i<subj; i++ )
  data[i].flg |= 0x2;

/****
     仮想拍手者全員が拍手を止めるまで，処理を継続する
****/
      while( sflg&0x1 ) {

if( *flg ) {
  memcpy( odata, odata+taplen2, size_s_C2 );
  bzero( odata+CLAPLEN2, size_s_t2 );

/* 拍手停止判定の閾値を，徐々に下げる */
  for( i=0, sumflg=0; i<subj; i++ )    sumflg += data[i].flg & 0x1;
  *clpps = (double)sumflg * invsubj;
  clpps3 = 1. - *clpps;
  thrshld2 = 0.95 - clpps3 * clpps3 * clpps3;

  for( i=0; i<subj; i++ )
    mkpoint( &data[i], *pitch );

  for( i=0; i<subj; i++ ) {
    if( data[i].flg&0x1 ) mksignal( odata, &data[i] );
    else if( data[i].flg&0x4 ) {
      mksignal( odata, &data[i] );
      data[i].flg &= 0xfb;
    }
  }

  shmctl( myshmid1, SHM_LOCK, &buf );
  memcpy( shmdata, odata, size_s_t2 );
  shmctl( myshmid1, SHM_UNLOCK, &buf );

/* 誰かまだ拍手を続けているのかどうかのチェック */
  for( i=1,sflg=0; i<subj; i++ )
    sflg |= data[i].flg;
/* もし，ユーザーが拍手をまた始めたら，リセットする */
  if( *pitch ) {
    *flg = 0;
    rstflg = 1;
    break;
  }
```

```
  *flg = 0;
}
      }

/****
     ユーザーが拍手を始めるまで，アイドリング
****/
      while( *pitch==0 ){};

/****
 改めて拍手を開始するための再初期化
****/
      if( !rstflg ) {
/* 拍手が全て止んでいる状態からのリスタート */
bzero( odata, size_s_d2 );
for( i=0; i<subj; i++ ) {
  data[i].pntr = (int) floor((double) taplen * drand48());
  data[i].mypch = (double)(*pitch<<shift) * data[i].coeff;
  data[i].flg = 0;
}
      } else {
/* 一部で拍手が残っている状態からのリスタート */
for( i=0; i<subj; i++ ) {
  if( !(data[i].flg &= 0x1) ) {
    data[i].pntr = (int) floor((double) taplen * drand48());
    data[i].mypch = (double)(*pitch<<shift) * data[i].coeff;
  }
}
      }
    }
  } /* while(1) の終了 */

/* ここへは来ません */
  shmdt( shmdata );
  shmdt( pitch );
  shmdt( prms );
  shmdt( clpps );
  free( odata );
  return( 0 );
}
```

D　SGI 用プログラム

HP-UX10.20 は，デフォルトではマルチスレッドには対応しておらず，pthread を使用するためには，別途 BUNDLE を購入する必要があった．そこで，付録 A~C に掲載したプログラムでは，vfork を用いて子プロセスとして幾つものプログラムを走らせることで，本システムを実現した．しかし，SGI のマシンであれば，デフォルトでマルチスレッドに対応しているため，セマフォを利用することにより，ひとつの実行ファイルで実現することが可能となる．また，アプリケーションによっては，そのほうが都合のよい場合もある．そこで，本システムをマルチスレッドを用いて，ひとつの実行ファイルで実現するためのソースコードを用意した．このソースプログラムは，HP で開発したプログラムを基にして書き直したものなので，基本的に多くの点で類似している．そこで，ここでは，それらのファイルの tar ボールのありかを示すに留める．なお，以下のアクセスポイントは，本稿執筆時 (2000 年 1 月 21 日) には存在しているが，数年のうちに抹消される可能性が高いことを付記しておく．

`http://www.mic.atr.co.jp/~ryou/App_sgi.tar.gz`

を取得して展開し，Applause ディレクトリにおいて，make をすると実行ファイル app が生成される．各ファイルおよびディレクトリの中身は，以下のとおりである．

appad.c A/D 関係のソースコード

appda.c D/A 関係のソースコード

appmk.c 拍手音生成関係のソースコード

clap48 拍手音の元となる音データファイル群

bump48 足音の元となる音データファイル群

利用にあたっては，以下の点に注意が必要である．

1. O2 では，内部雑音を大きいためか，ソースプログラム中の入力信号に対する感度を適切に変更しないと，うまく動作しない．
2. 実行されるマシンのスペックにも依るが，INDY では処理時間が間に合わない．
3. 附属のマイクロホンと内蔵スピーカーを使用すると，マイクとスピーカーの間の距離が近いため，スピーカーから出力された拍手音がマイクロホンに入力されてしまい，ピッチが速くなって拍手音が発散してしまう．
4. GUI のために，Motif のライブラリが必要となる．

E　HP 版「拍手喝采」の使用方法

先ず最初に，マイクロホンやスピーカーなどのオーディオ装置と DAXBOX を適切に接続する．マイクロホン (1 チャネル) は，DASBOX A/D モジュールの背面端子のどれかに接続する．一方，ヘッドホンあるいはスピーカー (2 チャネル：ステレオ) は，DASBOX D/A モジュールの背面端子のどれかに接続する．A/D，D/A に使用されるチャネルは，tempad.c および tempda.c 内において設定されるので，実際に接続したチャネル番号と

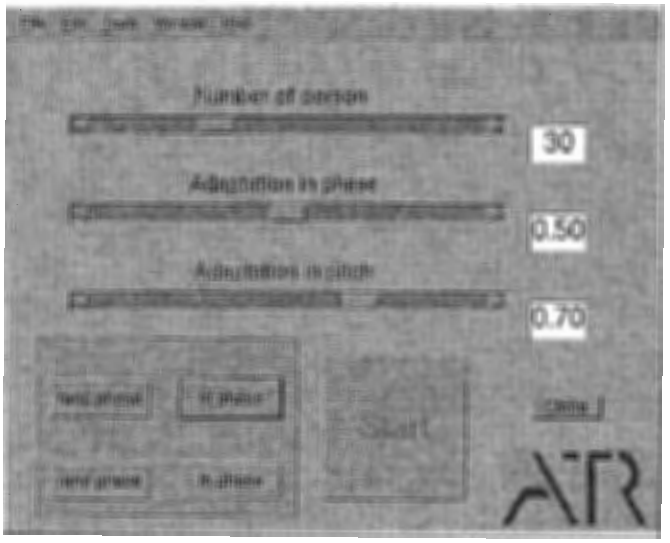


図 15: 本システムを制御するための GUI

対応するようにソースを書き換えてリコンパイルする．GUI は，MATLAB の Handle Graphics Object で製作しているため，MATLAB 上から起動する必要がある．起動すると，図 15 に示す GUI が表示される．

変更可能なパラメータは，

1. 拍手音の人数の最大値
2. ユーザー拍手のピッチへの追随性を制御するステップパラメータ μ
3. タイミングの同期を制御するステップパラメータ ν

の 3 つである．また，短縮ボタンとして，

1. 普通の拍手
2. 拍手から手拍子へ
3. まとまりのない足音
4. 行進のような足音

の 4 つの設定を用意しており，短縮ボタンを押すことで適当なパラメータが同時に選択される．μ, ν のスライドバーは，拍手が継続中でも変更可能だが，他のパラメータに関しては，拍手が止んでいる時に変更しなければならない．もし，拍手が継続している最中にこれらのパラメータを変更しようとすると，警告を示すウィンドウが現れる．

システムの動作開始/停止は，右下の Start/Stop のボタンで制御する．このボタンは，反転型になっており，“Start” と表示されている時にボタンを押すと，システムが動作を開始するとともに，このボタンが Stop ボタンに反転する．逆に，“Stop” と表示されている時にボタンを押すと，動作が停止するとともに，このボタンが Start ボタンに反転する．動作を停止するだけでなく，本システムを完全に終了する場合は，“close” ボタンを押すと，停止するとともに GUI も消える．